МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Навчально-науковий інститут новітніх освітніх технологій
Кафедра комп'ютерних наук

КОВБАСІСТА Юлія Вікторівна

Математичне та програмне забезпечення для оптимізації тестування web-орієнтованих систем / Mathematical Tools and Software for Optimizing of Web-Based Systems Testing

спеціальність: 121 - Інженерія програмного забезпечення
освітньо-професійна програма - Інженерія програмного забезпечення

Кваліфікаційна робота

Виконала студентка групи
ІПЗзм-21
Ю. В. Ковбасіста (Домчук)

Науковий керівник:
д.т.н., доцент А.В. Пукас

Кваліфікаційну роботу
допущено до захисту:

"___" _____ 20___ р.

Завідувач кафедри
_____ **А. В. Пукас**

ТЕРНОПІЛЬ - 2022

ВСТУП

Актуальність. Тестування є важливим кроком у розробці програмного забезпечення, оскільки, з одного боку, вимагає значних витрат на адреси, а з іншого боку, значно підвищує їх якість. Через теоретично доведену неможливість вичерпного тестування та високу потенційну вартість втрат через збої програмного забезпечення, існує потреба *в чітко визначеному та ефективному процесі тестування* , заснованому на збалансованих рішеннях, які приймаються по відношенню до тривалості та вартості тест для досягнення необхідного його рівня. впевненість у якості програмного забезпечення.

Процеси життєвого циклу програмного забезпечення» [5] всі дії зі створення програмного забезпечення (ПЗ) представлені у вигляді ряду процесів (основних, підтримки, процесів управління). Існує *інженерний, процесно-орієнтований підхід* до розробки програмного забезпечення. Цей підхід вимагає використання науково обґрунтованих кількісних вимірювань процесу тестування та результатів впровадження.У зв'язку з цим створення *моделей* і *методів кількісної оцінки* стану програмного забезпечення є важливим і актуальним завданням для програмної інженерії. побудувати на їх основі ефективний базовий процес тестування.

Сучасний стан розвитку програмної інженерії, застосування процесно-орієнтованого підходу до розробки програмного забезпечення дозволяє сформулювати декілька вимог до тестування програмного забезпечення, основними з яких є:

- підготовка до тестування, починаючи *з ранніх стадій* життєвого циклу програмного проекту;
- вибір і використання стратегій тестування, які відповідають не тільки цілям і рівням тестування, але й ресурсним обмеженням проектів;
- стандартизувати процес тестування з урахуванням вимог процесно-орієнтованого підходу;
- *кількісні критерії* виконання тесту.

Кількісні методи для завершення процесу тестування та керування ним за допомогою кількісних вимірювань все ще недостатньо використовуються в проектах розробки програмного забезпечення, що призводить до непередбачуваної якості та надійності. Ефективне впровадження процесу тестування може бути забезпечене лише за наявності достовірної та своєчасної інформації про стан програмного забезпечення, види ризиків та можливі втрати через збої.

Робочий зв'язок з науковими програмами, планами, темами

Напрямок дослідження безпосередньо пов’язаний з науковим напрямком кафедри «Комп’ютерні науки» Тернопільського національного економічного університету.

Мета і завдання дослідження

Метою роботи є розробка методу тестування веб - орієнтованих інформаційних систем.

Для досягнення поставленої мети поставлені наступні завдання:

1) Проаналізуйте основи тестування програмного забезпечення.

2) Провести порівняльний аналіз відомих методів і систем тестування.

3) Розробити принципи, методи та алгоритми тестування програмного забезпечення та його програмної реалізації;

4) Проаналізувати ефективність побудованого методу шляхом порівняльного тестування з відомими підходами.

5) Програмна реалізація розроблених методів і алгоритмів та проведення відповідного тестування.

Завдяки дослідженням ми розуміємо процеси, пов'язані з тестуванням веб-систем .

Предметом дослідження є методи та програмні засоби тестування веб - систем .

Методи дослідження

Теоретичні дослідження з розробки методів і програмного забезпечення для обробки слабко формальної інформації базуються на застосуванні

системного аналізу, методології функціонального моделювання, інженерії знань, семантичних мереж, теорії множин, теорії графів, теорії нечіткого висновку, теорії алгоритмів та об'єктно-орієнтованого. дизайн.

Новизна. Запропоновано метод тестування, який на своїй основі поєднує стратегії інтеграційного тестування тестування модулів/компонентів в одному додатку з використанням підходів, властивих функціональному тестуванню, тобто метод «сірого ящика») та системного тестування (тестування зовнішнього інтерфейсу) або інтерфейсів користувача.

РОЗДІЛ 1
АНАЛІЗ ПОТОЧНОГО СТАНУ ТЕСТУВАННЯ

1.1. Організаційна характеристика процесу тестування

тестування , один із методів подальшого підвищення якості програмних засобів розробки системи шляхом виявлення дефектів, які не були помічені іншими видами перевірок.

Під терміном інженерія тестування ми розуміємо частину інженерії програмного забезпечення , яка стосується всіх аспектів динамічного тестування програмного забезпечення.

Термін тест широко використовується в науковій літературі, але визначається по-різному.

Стандарт ANSI/IEEE Std. 610.12 [11] визначає термін тестування в найширшому розумінні як будь-яку діяльність з аналізу програми (статичне та динамічне тестування).

У книзі [12] Майєрс визначає цей термін у найвужчому сенсі: « тестування — це процес виконання програми (або її частини) з метою ідентифікації помилки ... " Налагодження - визначити точну причину відомих помилок і виправити їх».

Розділення цілей тестування та налагодження було першим кроком у розробці тестової техніки.

Хоча динамічне тестування стосується програмного забезпечення, воно завжди виконується в контексті комп’ютерної системи. У роботі використовується термін програмна система як частина програмного забезпечення, встановленого на комп’ютері.

програмною системою розуміють набір програмних засобів (одного або кількох), які функціонують у комп’ютерному середовищі та об’єднані бізнес-процесом обробки даних під час вирішення завдань користувача програмного забезпечення.

При розробці програмна система представлена через робочі продукти (документи, діаграми, код). Після закінчення розробки він отримує статус програмного продукту.

Відповідно до ДСТУ 3918-99 для контролю якості програмних продуктів (у тому числі всіх робочих продуктів) передбачені процеси верифікації та валідації.

Завдання верифікації та валідації полягає в тому, щоб перевірити та підтвердити , що кінцевий програмний продукт буде виконувати своє призначення та задовольняти користувачів. Ці процеси контролюють якість шляхом виявлення помилок у продуктах процесів LC, не досліджуючи причини появи цих помилок [13].

чи правильно розробляється програмний продукт (наприклад, чи правильний код програмного модуля для специфікації проекту цього модуля), досліджуючи перетворення деяких робочих продуктів (вхідних даних) у. інші (вихідні) продукти праці .

Метою валідації є перевірка набору робочих продуктів, отриманих на певному етапі процесу розробки, щоб переконатися, що вони належним чином розроблені , тобто відповідають меті та початковим вимогам, визначеним для програмного продукту (наприклад, The набір програмних модулів фактично представляє необхідний програмний продукт, набір фактично виконаних тестів достатній для перевірки всіх функцій програмного забезпечення тощо).

Ці процеси взаємопов'язані і зазвичай визначаються єдиним терміном «верифікація та валідація», що відповідає терміну « Верифікація і Перевірка» (V&V), що використовується в зарубіжній літературі. Процеси V&V повинні здійснюватися, починаючи з найраніших етапів фази розробки програмного забезпечення та по всіх робочих продуктах розробки (включаючи, наприклад, креслення).

Слід зазначити, що, з одного боку, тестування є одним з основних процесів розробки програмного забезпечення, а з іншого боку, воно вважається основним методом V&V . На нашу думку, тестування, як процес, недостатньо чітко

визначено в діючих стандартах, оскільки ефективна реалізація процесу тестування вимагає виконання певних дій майже на кожному етапі життєвого циклу проекту та в багатьох процесах, наприклад , у процесах, пов’язаних із визначенням вимог, забезпеченням якості та V&V . Результати дослідження процесів і методів V&V представлені в роботах [2, 14].

Такі поняття, як «помилка», «несправність», «відмова», «проблема», «аномалія» тісно пов'язані з тестуванням, щодо визначення якого в літературі досі точаться суперечки. Ці поняття визначаються по-різному не лише в науковій літературі з якості та надійності програмного забезпечення, але й у стандартах.

Зокрема, стандарт ANSI/IEEE-729-83 [15] дає два визначення відмови:

1) відмова - нездатність комп'ютерної системи або її компонента виконувати необхідні функції в заданих межах;

2) збій - це відхилення програми від роботи, визначеної вимогами програми.

Поєднуючи два визначення, термін відмова вживається в магістерській роботі в такому значенні:

збій програмного забезпечення (збій) , що полягає у переході програмного забезпечення з працездатного стану в неробочий стан або одержанні результатів поза межами допустимих значень.

Збої можуть бути обумовлені як зовнішніми факторами (відмовами елементів операційного середовища, в тому числі користувача системи), так і внутрішніми факторами - дефектами програмного забезпечення.

Дефект (дефект) програмного забезпечення - це запис елемента програми (коду) або тексту документа (продукту роботи), використання якого може призвести до некоректного представлення цього елемента комп'ютера (помилка (несправність) .) або особою (помилка виконавця (помилка).

Помилки розробника завжди призводять до дефекту в будь-якому з процесів розробки. Дефекти можуть мати ТУ, початкові або проектні ТУ, кодові тексти, експлуатаційну документацію тощо.

Недоліки в одному з процесів розробки програмного забезпечення можуть призвести до помилки, змусивши когось неправильно витлумачити початкову інформацію та прийняти неправильні рішення.

Не всі дефекти програмного забезпечення можуть спричинити збій. Не всі збої можуть бути наслідком дефекту програмного забезпечення або збою середовища (помилковий збій, наприклад, через дефекти в тестах, апаратний збій). Будь-який збій (як подія) супроводжується появою аномалії (проблеми) - зовнішнього прояву помилок і дефектів.

Дефекти в програмі, які не були виявлені в результаті перевірки коду, є джерелом потенційних помилок і програмних збоїв. Вираження несправності у вигляді відмови залежить від ситуації, в якій буде працювати користувач (чи буде обраний шлях, що містить помилковий код), і вхідних даних.

Під час тестування, якщо виявлено відхилення продуктивності від очікуваних, таке відхилення встановлюється як «проблема» до з’ясування її причин.

Результати дослідження взаємозв'язку цих понять наведені в роботах [2,16].

динамічно перевіряти поведінку програми на кінцевому наборі тестових даних, спеціально відібраних із нескінченного простору введення , щоб відповідати встановленій очікуваній поведінці [17].

Слова, виділені курсивом, є критичними для тесту.

Динамічний: тестування – це завжди виконання програми .

Готово: навіть для невеликої програми теоретично можливо створити стільки тестів, що це може зайняти роки [18, 19]. Однією з головних проблем тестування є неповнота, оскільки на практиці весь набір тестів можна вважати нескінченним. Кількість тестів, які можна зробити за обмежений час, становить прибл. Таким чином, тестування завжди передбачає певний «компроміс» між обмеженими термінами і потенційно нескінченною кількістю тестів. Це створює добре відомі проблеми тестування, такі як прийняття рішень щодо адекватності

тесту, і проблеми управління, пов’язані з оцінкою витрат (вартість, час, персонал) на тестування.

Вибране : проблема вибору обмеженого набору тестів пов’язана з проблемою адекватності тесту. Загалом методи тестування відрізняються підходом до вибору набору тестових даних із вхідного простору.

Через неможливість вичерпного тестування були розроблені різні методи для зменшення кількості тестів і пошуку критеріїв адекватності тесту [19, 20, 21].

Очікування: Ви повинні вміти (хоча це не завжди легко) визначити, чи є результати виконання програми правильними, чи відповідає спостережуване виконання очікуванням або специфікаціям користувача. У літературі з тестування це називається задачею оракула (бенчмарка), для вирішення якої можуть використовуватися різні підходи (оцінка результатів, порівняння з існуючим бенчмарком, узгодження з користувачем тлумачень понять «несправності»). і «невдача»).

Розподіл тестових дій зазвичай представляється окремими процесами у вигляді V-подібної моделі [6, 23], яка показує процеси декомпозиції та інтеграції програмного забезпечення та відповідні рівні та дії динамічного тестування (рисунок 1.1).).

Але такий розподіл дій тестування через різні процеси не дозволяє визначити ролі учасників процесу тестування на кожному етапі розробки ПЗ, розрахувати вартість підготовки та проведення тестування на кожному рівні. Для ефективного виконання завдань тестування, що виконуються на різних етапах розробки, необхідно об'єднати їх в єдиний процес тестування.

Розділивши тестові дії, розподілені між різними процесами НС, в один базовий процес, можна буде створити середовище та заздалегідь визначити тестові ресурси , встановити обсяг і час тестування в рамках плану проекту програмного забезпечення. .

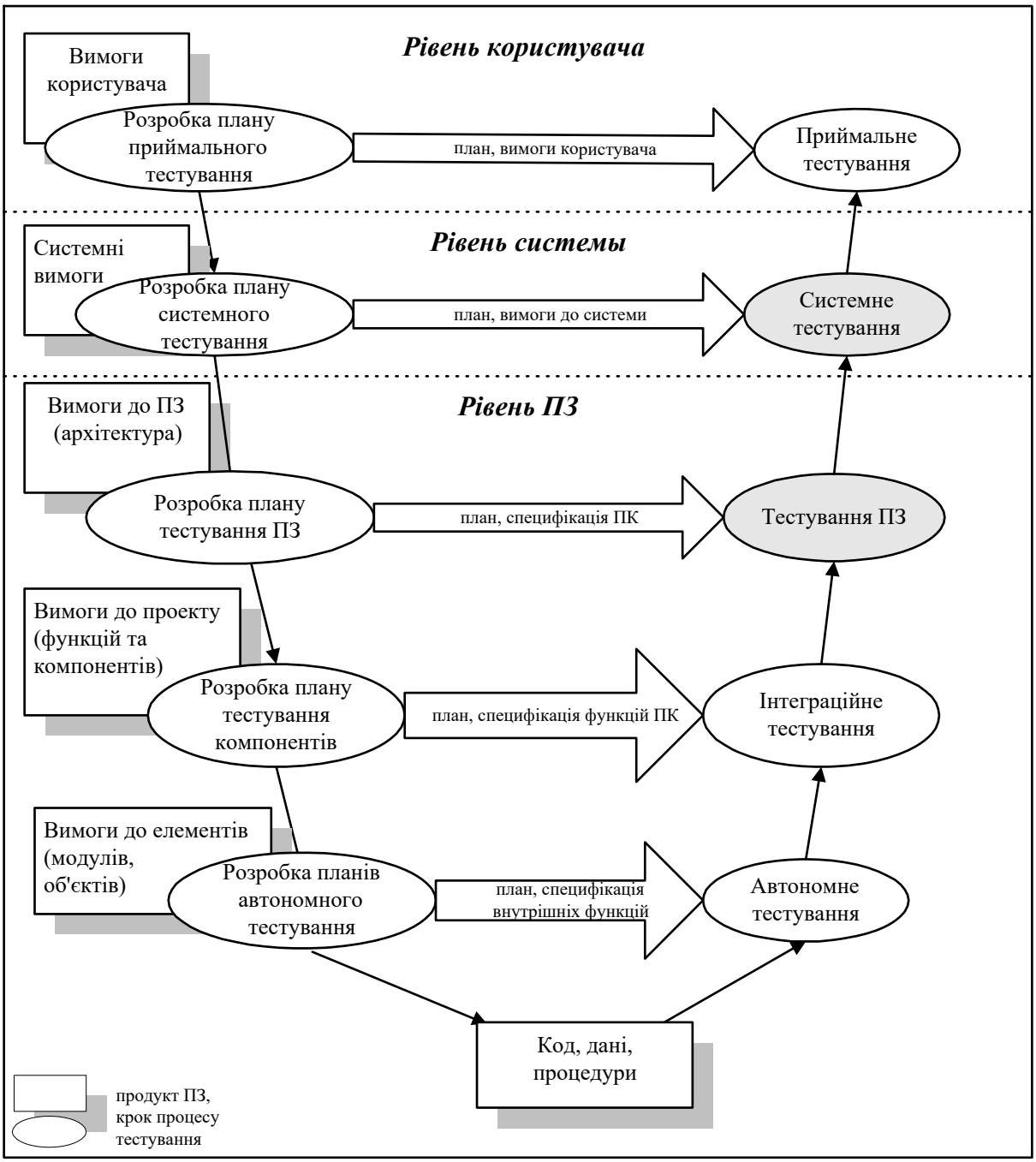

Рис. 1.1. V-подібна модель житлового центру з «вмонтованими» тестовими завданнями.

1.2. Показники та критерії проходження тесту

Вимірювання мають велике значення в розробці програмного забезпечення, оскільки їх використання дозволяє отримати об'єктивну

інформацію про стан програмного забезпечення та процеси розробки. Рекомендований склад метрик наведено в стандарті ISO/IEC 9126 [28].

Оскільки основними процесами, які зазвичай досліджуються в розробці якості та надійності, є процеси «введення помилок», «усунення дефектів» і «процес відмови», вважається, що «кількість дефектів», «щільність дефектів» є основним показником . » і «серйозність відмови».

Кількість дефектів, виявлених під час тестування, не можна вважати хорошим індикатором якості програмного забезпечення, вони більше служать індикаторами ефективності тестування [29]. При цьому невелика кількість дефектів, виявлених під час тестування, не є показником поганого тестування, а може відображати ефективність процесу розробки. Коефіцієнт є об'єктивним показником ефективності тесту:

$$E_e = \frac{D_e}{D_e + D_o}$$

для E_e ефективність тесту, D_e кількість дефектів, виявлених під час тестування, D_o - кількість несправностей, виявлених під час експлуатації.

На жаль, за цим показником неможливо контролювати процес тестування, оскільки величина D_o Може бути надто пізно.

Реєстрація дефектів повинна бути однією з цілей процесу тестування, вона також визначається як обов'язкова вимога базового стандарту ISO 9000-3. Реєстрація дефектів служить основою для формування історичних довідкових банків даних про проекти програмного забезпечення, які можуть бути використані для вдосконалення процесів розробки програмного забезпечення. Аналізуючи дані щодо щільності дефектів програмних продуктів, вдалося визначити середню щільність дефектів – 6 дефектів на 1000 рядків для США та Європи, 2 – для Японії [2].

На додаток до цих основних показників, такі показники корисні для вимірювання результатів тестування:

1) метрики для оцінки набору тестів за вибраними критеріями охоплення, такі як метрики охоплення для вимог до тестів, кодів або функцій;

2) метрики тенденції дефектів (знайдено, усунено, за серйозністю, пріоритетом тощо).

Крім того, для ефективного керування процесом тестування важливими показниками є час і вартість тестування.

Практичне використання метрик для управління процесом тестування має багато проблем, а саме:

- зібрати вихідні дані для розрахунку метрики;
- інтерпретація результатів розрахунків і ступінь довіри до них;
- як використовувати метрики для управління тестуванням на всіх рівнях;
- вибір мінімального набору метрик, прийнятних у контексті конкретного процесу тестування.

Показники, які будуть включені в процес тестування, слід вибирати так, щоб вони виступали в якості об’єктивних індикаторів стану виконання процесу тестування та поточного стану програмного забезпечення.

Через неможливість вичерпного тестування однією з найважливіших проблем управління проектами є прийняття обґрунтованого рішення щодо завершення тестування та випуску програмного забезпечення.

Об'єктивні критерії виконання тесту базуються на кількісних вимірюваннях. Існуючі підходи до створення кількісних критеріїв можна розділити на чотири категорії:

1) критерії на основі показників охоплення;
2) критерії, засновані на тяжкості дефектів;
3) критерії, засновані на оцінках серйозності відмови та надійності;
4) вартість, час випробувань і критерії оптимізації надійності.

У першому підході критерії встановлюються шляхом визначення допустимих граничних значень метрик покриття (вимог, функцій, коду). За показниками покриття ці критерії поділяються на структурні та функціональні. Випробування припиняється при досягненні встановлених граничних значень. Ці критерії є показником повноти проведеного тестування, але не враховують час, витрачений на тестування.

Критерії другої категорії засновані на розподілі виявлених дефектів за ступенем серйозності і встановленні граничних значень для усунення кількості дефектів з урахуванням їх серйозності. Відповідно до цього підходу випробування припиняють після усунення всіх найбільш серйозних дефектів і визначення кількості дефектів інших видів. Ці критерії відображають не сам процес тестування (виявлення дефектів), а процес усунення недоліків розробника. Вони не враховують час, витрачений на тестування та виправлення дефектів.

У критеріях третьої категорії, заснованих на оцінках серйозності відмови та надійності, тестування триває до досягнення граничних значень, заснованих на показниках надійності (серйозності відмови, середнього часу роботи без відмови або ймовірності безвідмовної роботи). Ці критерії не враховують наслідки відмов для користувача, а лише їхню ймовірність [30].

До критеріїв цієї категорії відносяться критерії, засновані на щільності дефектів, які встановлюють граничні значення щільності дефектів програмного забезпечення. Використання цих критеріїв вимагає розрахунку розміру початкового програмного коду та знання його внутрішньої структури [2].

Критерії оптимізації (четверта категорія) спрямовані на пошук «компромісу» між часом тестування та вимогами до якості. У цьому підході один із атрибутів вибирається для цільової функції, наприклад, час, вартість або надійність, а інші діють як обмеження. Огляд основного виду критеріїв цієї групи, заснований на використанні моделей надійності, наведено в роботах [27]. Розглянемо типове завдання, щоб визначити найкращий час тестування.

Нехай C_1 — вартість усунення однієї відмови під час розробки та тестування; C_2 – вартість усунення відмови в процесі експлуатації, $C_2 > C_1$; C_3 – вартість одиниці тестового часу (наприклад, один день), t_e – тестовий час, $\lambda(t_e)$ - *серйозність програмних збоїв під час* роботи за умови проведення тесту протягом *часу* ; $\mu(t_e)$ функція підвищення надійності.

Тоді загальна вартість тестування та усунення дефектів у процесі експлуатації за умови тестування ПЗ у момент часу $t_{e.}$ - розмір:

$$D(t_e) = D_1(t_e) + D_2(t_e) + D_3(t_e)$$

де $D_1(t_e) = C_1 \mu(t_e)$ - витрати на усунення відмов під час випробувань;

$D_2(t_e) = y \lambda(t_e) C_2$ - витрати, пов'язані з пошуком та усуненням відмов під час експлуатації, де y - очікуваний час роботи;

$D_3(t_e) = t_e C_3$ - витрати на тестування.

Необхідно знайти такий час випробувань t^*, при якому сумарні витрати на усунення дефектів під час випробувань і експлуатації будуть мінімальними, *т.е.*

$$t_e^* = \{t: D(T/t) \geq D(T/t_e^*),\ 0 \leq t \leq t_{max}\},$$

де $t_{макс}$ - максимальний дозволений час тестування.

Оскільки модульна структура програмного забезпечення тестується на різних рівнях і охоплює різні види тестування, рекомендується використовувати комплексні критерії для тестування. Один із таких критеріїв запропоновано в [15]. За цим критерієм можна перевірити:

- пройдено всі заплановані функціональні випробування;
- структурне тестування було виконано з набором тестів, які забезпечують 100% охоплення рядків коду , 80% охоплення логічних умов і 100% охоплення викликів процедур;
- інтенсивність виявлення відмов не перевищує 40 нових відмов за 1000 годин випробувань;
- тривалість безперервної роботи ПЗ без збоїв досягає 100 разів.

В умовах обмежених ресурсів для тестування критерій завершення повинен бути встановлений на основі *оцінки ризику* програмного збою. Необхідно враховувати, які ризики були виявлені, які були усунені в результаті тестування, і серйозність дефектів, що залишилися.

1.3. Види тестування ПЗ

Залежно від переслідуваних цілей види тестів можна умовно розділити на такі види:

1. Функціональний
2. Не застосовується
3. Що стосується змін

Функціональні тести базуються на функціях і характеристиках, а також взаємодії з іншими системами, і можуть бути представлені на всіх рівнях тестування: компонент або модуль (Component/Unit testing), інтеграція (testing testing), система (системи тестування) і прийняття . (Приймальні випробування)). Типи функціональних тестів оцінюють зовнішню поведінку системи. Деякі з найпоширеніших типів функціональних тестів перераховані нижче:

- Функціональне тестування (Functional testing)
- Тестування безпеки та контролю доступу
- Тестування сумісності

Нефункціональне тестування описує тести, необхідні для визначення характеристик програмного забезпечення, які можна виміряти різними величинами. Загалом, це перевірка «Як» працює система. Нижче наведено основні типи нефункціональних тестів:

- Всі види тестування продуктивності:
  - тестування продуктивності та навантаження
  - стрес-тест (Stress Test)
  - тест на стабільність або надійність (Stable / Reliability Test)
  - об'ємний тест (Volume test)
- Тест установки (тест установки)
- Тестування зручності використання
- Тестування на відмову та відновлення
- Тест конфігурації (Configuration Test)

Після внесення необхідних змін, таких як виправлення помилки/несправності, програмне забезпечення слід протестувати повторно, щоб підтвердити, що проблему справді вирішено. Нижче наведено типи тестування, які необхідно виконати після встановлення програмного забезпечення, щоб підтвердити працездатність програми або правильність виправлення помилок:

- Тест на дим (Smoke Test)
- Регресійне тестування
- Побудова перевірочного тесту
- Санітарний тест або перевірка консистенції / працездатності (Sanitary Test)

Тестування безпеки (Тестування безпеки та контролю доступу) . Стратегія тестування, яка використовується для перевірки безпеки системи, а також аналізу ризиків, пов’язаних із забезпеченням цілісного підходу до захисту програм, хакерських атак, вірусів, несанкціонованого доступу до конфіденційних даних. Тестування безпеки може бути автоматизованим або ручним, включаючи перевірку позитивних і негативних тестів. Він базується на трьох основних принципах - конфіденційність, цілісність і доступність (confidentiality, integrity, availability)

Конфіденційність. Конфіденційність - це акт приховування певних ресурсів або інформації. Можна розуміти, що конфіденційність обмежує доступ до ресурсу певної категорії користувачів, або, іншими словами, умови, за яких користувачеві дозволений доступ до цього ресурсу.

Цілісність. Існує два основні критерії визначення поняття доброчесності:

1. Довіра. Очікується, що ресурс буде змінено відповідним чином лише певною групою користувачів.

2. Втрата і відновлення. У випадку, якщо авторизований або неавторизований користувач знищує або неналежним чином змінює дані, вам необхідно визначити, наскільки важливою є процедура відновлення даних.

Доступність - це вимога, згідно з якою ресурси повинні бути доступні для авторизованого користувача, внутрішнього об'єкта або пристрою. Зазвичай чим важливіший ресурс, тим вищий рівень доступності.

Тестування сумісності . З розвитком мережевих технологій та Інтернету взаємодія різних систем, сервісів і додатків є дуже важливою, оскільки будь-які супутні проблеми можуть призвести до падіння репутації компанії, що призведе до фінансових втрат. Тому до тестування взаємодії слід підходити з усією серйозністю.

Тестування сумісності — це функціональний тест, який перевіряє здатність програми взаємодіяти з одним або декількома компонентами або системами та включає тестування сумісності та тестування інтеграції.

Програмне забезпечення з хорошими характеристиками сумісності можна легко інтегрувати з іншими системами без будь-яких серйозних модифікацій. У цьому випадку кількість змін і час, необхідний для їх виконання, можна використовувати для вимірювання сумісності.

Навантажувальне тестування (Види тестування продуктивності). Навантажувальне тестування включає наступні тести:

Тестування продуктивності . Завдання тесту продуктивності полягає в тому, щоб визначити масштабованість програми під навантаженням, і відбувається наступне:

- 648/2012 Текст, що стосується вимірювання ЕЕА часу виконання вибраних операцій за певної інтенсивності виконання цих операцій
- який визначає кількість користувачів, які одночасно працюють з додатком
- визначити межі допустимої продуктивності при збільшенні навантаження (при збільшенні інтенсивності цих операцій)
- працездатність при підвищених навантаженнях, екстремальному стані, стресах

Тестування навантаження та продуктивності . В англійській термінології також можна зустріти ще один вид тесту - Load Test - тестування реакції системи

на зміну навантаження (в межах допустимого). Ми побачили, що Load і Performance досі мають ту саму мету: перевірити продуктивність (час відгуку) за різних навантажень. Тому ми з ними не розлучалися. При цьому хтось може поділитися. Головне - розуміти цілі того чи іншого виду тестування і намагатися їх досягти.

Стрес-тест (Stress Test) . Стрес-тестування дозволяє перевірити, наскільки функціональні додаток і система в цілому в стресових умовах, а також оцінити здатність системи до регенерації, тобто повернення в нормальний стан після припинення стресу. Стрес у цьому контексті може підвищити інтенсивність операцій до дуже високих значень або екстрену зміну конфігурації сервера. Також одним із завдань стрес-тесту може бути оцінка погіршення продуктивності, тому цілі стрес-тесту можуть збігатися з цілями тесту продуктивності.

Об'ємний тест (Volume test) . Завдання об’ємного тестування полягає в тому, щоб отримати оцінку продуктивності, коли обсяг даних у базі даних програми збільшується, при цьому відбувається:

- 648/2012 Текст, що стосується вимірювання ЕЕА часу виконання вибраних операцій за певної інтенсивності виконання цих операцій
- можна визначити кількість користувачів, які одночасно працюють із програмою

Тестування стабільності або надійності (Stable / Reliability Testing) . Завдання тестування на стабільність (надійність) — перевірити працездатність програми під час тривалого (багатогодинного) тесту із середнім рівнем навантаження. Час виконання операцій може відігравати другорядну роль у цьому типі тестування. При цьому на перше місце виходить відсутність витоків пам'яті , перезавантаження серверів під навантаженням та інших аспектів, що впливають на стабільність роботи.

Тестування інсталяції спрямоване на перевірку успішної інсталяції та конфігурації, а також оновлення або видалення програмного забезпечення. На даний момент найбільш поширена установка програмного забезпечення за

допомогою інсталяторів (спеціальних програм, які самі також вимагають відповідного тестування, опис яких було розглянуто в статті « Особливості тестових інсталяторів »).

У реальних умовах інсталяторів може не бути. У цьому випадку вам доведеться самостійно встановлювати програмне забезпечення, використовуючи документацію у вигляді інструкцій або файлів readme, покроково описуючи всі необхідні дії та перевірки.

У розподілених системах, де додаток розгортається у вже запущеному середовищі, простого набору інструкцій може бути недостатньо. Для цього часто пишеться план інсталяції (Deployment Plan), який включає не тільки кроки для встановлення програми, а й кроки для повернення до попередньої версії, у разі невдачі. Сам план встановлення також повинен пройти процедуру тестування, щоб уникнути проблем під час його фактичного впровадження. Це особливо актуально, якщо інсталяція виконується на системах, де репутація та багато грошей втрачаються в кожну мить простою, наприклад: банки, фінансові компанії чи навіть банерні мережі. Тому інсталяційне тестування можна назвати одним із найважливіших завдань у забезпеченні якості програмного забезпечення.

Юзабіліті-тестування – це метод тестування, спрямований на ступінь зручності використання, зручності навчання, зрозумілості та привабливості для користувачів розробленого продукту в контексті заданих умов. [ISO 9126]

Юзабіліті-тест оцінює рівень зручності використання програми за наступними пунктами:

- продуктивність, ефективність (ефективність) - скільки часу і кроків знадобиться користувачеві для виконання основних завдань програми, наприклад, розміщення новин, реєстрація, покупка тощо? (Чим менше, тим краще)
- точність - скільки помилок припустився користувач під час роботи з додатком? (Чим менше, тим краще)

- активація в пам'яті (recall) - скільки користувач пам'ятає про програму після припинення роботи з нею протягом тривалого часу? (Відновити роботу після перерви має бути швидше, ніж з новим користувачем)
- емоційна реакція - що відчуває користувач після виконання завдання - розгубленість, пережитий стрес? Чи буде користувач рекомендувати систему своїм друзям? (Краще позитивна реакція)

Тест на дим (Smoke Test) . Концепція димового тестування прийшла з інженерного середовища. При встановленні нового «заліза» перевірка вважалася успішною, якщо з установки не йшов дим. У сфері тестування програмного забезпечення спрямована на проведення поверхневої перевірки кожного програмного модуля на працездатність і наявність критичних і блокуючих дефектів, які швидко виявляються. Димове тестування зазвичай виконує сам програміст, немає сенсу відправляти програму, яка не пройшла цей тест, на більш поглиблене тестування. Для полегшення роботи, економії часу та людських ресурсів рекомендується автоматизувати димові випробування.

Регресійне тестування . Регресійне тестування — це цикл тестування, який відбувається, коли вносяться зміни під час тестування системи або фази підтримки продукту. Основною проблемою регресійного тестування є вибір між повним і частковим повторним тестуванням і заміною набору тестів. При частковому повторному тестуванні контролюються лише ті частини проекту, які стосуються змінених компонентів. У GMP це шляхи, які містять модифіковані вузли, і, як правило, це методи та класи, які за рівнем були вище модифікованих, але включають їх у свій контекст

Величезна кількість тестів, які характеризують етап тестування системи, можна пропустити без втрати показників якості продукту лише за допомогою регресійного підходу.

Приклад тесту регресії. Отримавши звіт про помилку, програміст аналізує вихідний код, знаходить помилку, виправляє її та тестує модуль або інтеграцію результату.

Далі тестувальник, перевіряючи внесення змін програмістом, повинен:

- Перегляньте та схвалюйте виправлення помилок. Для цього необхідно виконати тест, зазначений у звіті, за допомогою якого виявлено помилку.
- Спробуйте відтворити помилку іншим способом.
- Заперечення проти наслідків виправлень. Ці виправлення могли спричинити помилку (дану помилку) у коді, який раніше працював нормально.

Перевірка Тест Конструкція . Тест має на меті визначити, чи відповідає випускна версія критеріям якості для тестування. За своїми цілями він є аналогом димового тесту, спрямованого на прийняття нової версії для подальшого тестування або експлуатації. Це може піти далі, залежно від вимог до якості випущеної версії.

Санітарний тест або перевірка консистенції / працездатності (Sanitary Test) . Вузького тестування достатньо, щоб довести, що певна функція працює відповідно до вимог, викладених у специфікації. Це підмножина регресійного тестування . Він використовується для визначення працездатності певної частини програми після внесення змін до неї або навколишнього середовища. Зазвичай робиться вручну.

Різниця між випробуванням на санітарні умови та випробуванням на дим (санітарія проти випробування на дим)

Деякі джерела помилково вважають, що санітарія та тестування на дим – це одне й те саме. Вважаємо, що ці види тестування мають «вектори руху», спрямовані в різні боки. На відміну від димового тестування, перевірка працездатності глибоко зосереджена на функції, що перевіряється, а димове тестування має широку спрямованість, щоб охопити якомога більше функціональних можливостей тестами за найкоротший проміжок часу.

1.4 . Аналіз основних методів тестування веб - інформаційних систем

Загальна ідеологія тестування орієнтована на використання двох основних підходів – тестування в режимі «білого ящика» або «чорного ящика».

У першому випадку розробник системи виконує процес тестування з використанням вихідного коду. У другому — готовий додаток або (його модулі) тестується лише на основі виконуваного коду системи. Існує проміжний підхід, який використовує ідеї перших двох – тестування «сірого ящика», коли тестувальнику частково або повністю надається вихідний текст системи, а також її виконуваний код. Такий підхід дозволяє виконувати всі процедури тестування програмного забезпечення на найвищому рівні.

Однак під час тестування реальних програм може не вистачити часу чи інших ресурсів для визначення та реалізації повного набору тестових випадків. У такому випадку необхідно визначити, які тести є найважливішими, а які функції програмного забезпечення для аналізу не перевірятимуться.

Зазвичай тестер в реальних умовах дотримується підходу тестування методом «чорного ящика». Потім зроблено короткий аналіз цих двох найпоширеніших методів тестування - «інкрементного підходу» [1-4] і «схематичного підходу» [5]. Обидва методи відповідають функціональному тестуванню ПЗ.

Метод інкрементального підходу (докладно описаний в [3]) використовується у випадку, коли тестувальник, який має виконуваний код системи, проходить кілька етапів тестування. Вивчення програми та ознайомлення з її функціями є важливим етапом навчання. Тестувальники створюють тести на льоту, часто під впливом попередніх тестів. Цей підхід, який називається пошуковим тестуванням, є першим кроком у процесі прийняття рішення про те, що тестувати.

Недоліки: результати, видані додатком, можуть вплинути на рішення тестувальника - він може вважати отримані результати правильними; коли тестер стикається з помилкою, може статися, що немає запису послідовності вхідних параметрів, дій користувача тощо, щоб визначити правильний шлях у попередньому запиті чи стані. Це може ускладнити відтворення помилки.

Базове тестування - створення базового тесту. Зазвичай базовий тест простий: він базується на найпростішому шляху в програмі, який використовує налаштування за замовчуванням або інші прямі вхідні дані. Враховуючи задані вхідні дані, тестер повинен визначити результуючі дані (будь-які спостережувані вихідні дані або відсутність таких даних).

Недоліки: відсутність точного представлення правильних даних; необроблені дані не можуть гарантувати успіх навіть у базовому тестуванні. Аналіз трендів є необов'язковим кроком, на якому природа поведінки відстежується шляхом зміни значення однієї конкретної змінної. Навіть якщо результат справжніх вихідних даних точно невідомий, можна судити про те, чи змінюються значення вихідних даних в очікуваному напрямку.

Інвентаризація – складається з визначення типів даних, доступних у програмі, з наступною класифікацією стану, у якому може перебувати кожен елемент даних. Тест гарантує, що кожен стан використовується принаймні один раз. Кожен набір станів визначає список інвентарю.

Недоліки: після того, як розробники вирішать проблему та створять нову версію програми, тестувальники мають повторно запустити оригінальні тести.

об'єднання предметів інвентарю, щоб можна було визначити, чи очікується, що вони будуть працювати разом. Не всі списки інвентарю містять прості дані або значення статусу.

Недоліки: при визначенні кожної комбінації виходить велика кількість тестів - набагато більше, ніж можна було б зробити за прийнятний час.

Граничні оцінки – це дослідження меж програмних компетенцій. Межі (або обмеження) можливостей програми визначаються даними. Обмеженнями можуть бути мінімальні та максимальні значення в діапазоні даних; мінімальний і максимальний розмір поля тощо. Загальне емпіричне правило полягає в тому, щоб створити три тести для охоплення таких значень: порогове значення; граничне значення -1; граничне значення +1.

Неправдиві дані - спроба відключити програму, яка створює умови, які, ймовірно, не працюватимуть для звичайного користувача. Мета полягає в тому,

щоб перевірити прийнятну поведінку програми, навіть якщо тестові дані не описують типову ситуацію. Створення тесту з помилковими даними зазвичай призводить до повідомлення про помилку. Очікується, що програма візьме ці дані та повідомить користувача про помилку.

Створення напруги. На цьому етапі тестувальники відходять від функцій додатків і оцінюють умови їх реалізації з точки зору робочих характеристик і відновлення системи після збоїв. Навіть якщо система перезавантажиться, після перезавантаження все має працювати належним чином.

Резюме: Метод інкрементального підходу (еволюційний підхід) може бути використаний як основа для тестування будь-якої програми. Однак метод не враховує характеристики програми. Метод піддається «автоматизації» на тому етапі, коли відомі всі необхідні тестові завдання (test case). Це складно при виконанні інтеграційного тестування через те, що необхідно «охопити» всі можливі комбінації (записи списку інвентаризації) вхідних параметрів і станів, отримані з «видом» взаємодії кожного модуля системи. Тестування додатків «комплексно» вимагає багато часу.

Схематичний підхід – це метод, що використовується для аналізу вимог [5]. Вимоги перетворюються на діаграми для переліку різних умов тестування. Схематичний метод контролює процеси і не залежить від вмісту системи. Часто це дозволяє знайти інформацію, якої немає у вимогах.

Кожен елемент у схемі остаточно визначатиме вхідні стани для тестового випадку. Набір станів системи можна перетворити на набір функцій тощо. Схема, як правило, є деревоподібною структурою (графом), в якій існує однозначно визначений шлях між коренем і кожним листом. Цей шлях встановлює певний набір вхідних станів, які потім використовуватимуться для визначення тесту.

Схема містить вузлові точки (або точки розгалуження), які представляють покажчики, типи значень або вхідні змінні. Нумерація вузлів вказує на конкретний вхідний стан відповідного предка. Кількість листків на дереві (або

шляхів на діаграмі) дає приблизну кількість тестів, необхідних для тестування всіх функцій програми.

Створення схеми є ітеративним процесом. Допомагає остання версія схеми завершити тестові випадки. Кожен шлях на діаграмі (від кореня до аркуша) визначає вхідну комбінацію, яка є основою для тесту.

Процес тестування розділений на послідовні кроки та підтримує різні рівні [6,7]: поелементне тестування (впливає на найменші одиниці компіляції будь-якої програми); тестування рівня інтеграції (комбінування окремих елементів і підтвердження їх правильного функціонування); тестування системного рівня (застосовується до цілих додатків, часто відноситься до поведінки користувача під час роботи з системою). Результати перевірки заносять у таблицю. На основі цих даних оцінюють тестове покриття структури програми.

Підсумок: таблиці використовуються протягом усього процесу тестування, починаючи з поелементного тестування і закінчуючи тестуванням на рівні системи. Опис програми легко перетворюється за допомогою діаграми переходу в еквівалентну таблицю станів, з якої можна легко отримати тестові приклади. Для програм, де використовується велика кількість складних комбінацій даних, тестові таблиці можуть посилатися на файли, що містять фактичні описи вхідних даних і вихідних результатів. За допомогою таблиць рішень можна відстежувати складні комбінації умов і пов’язані з ними дії.

Тестування веб-додатків і баз даних. Додатки на базі технологій WWW створюють нові виклики для розробки та тестування [4, 8, 9]. Тести для веб-вузла повинні зосереджуватися на очікуваній поведінці вузла. Необхідно розглянути такі проблемні моменти: функціональні можливості; практичність; навігація ; форма, зміст сторінки. Принципи тестування веб-додатків детально описані в [4].

Тестування бази даних часто є дуже важливою частиною тестування додатків [10]. При тестуванні баз даних потрібне всебічне знання тестованої програми. Ключові проблеми, які виникають під час тестування баз даних, включають цілісність даних; достовірність даних (форма при введенні в бази даних); маніпулювання та оновлення даних.

Загальне резюме: Тестування розподіленої програми є досить складним завданням. Якщо програма використовує багаторівневу клієнт-серверну архітектуру, підходи, описані вище, слід застосовувати до кожного рівня.
Але в той же час вам потрібно провести інтеграційне тестування, щоб перевірити інтерфейси системи, і тестування системи, щоб перевірити зовнішні інтерфейси (зазвичай інтерфейси користувача або зовнішню систему).
Зазвичай для складних додатків тест поділяють на покрокове тестування окремих компонентів, окремих модулів, окремих підсистем, окремих частин додатка; інтеграція - тестування взаємодії кожної частини програми, інтерфейсу користувача, інтерфейсу системи в цілому.

Загального методу немає, тому що потрібно використовувати «особливості» програми. Ефективного тестування неможливо досягти без знань вихідний код (структура) всіх компонентів системи. Особливо ускладнюється завдання, якщо система розробляється окремими частинами або в систему інтегруються «чужі» модулі.

1.5 _ _ Постановка проблеми дослідження

Інформаційні системи на базі технологій WWW створюють нові проблеми, як для розробки, так і для тестування [1]. Тести для веб-додатків повинні зосереджуватися на прогнозуванні поведінки вузла. Необхідно розглянути такі проблемні моменти: функціональні можливості; практичність; навігація ; форма, зміст сторінки. Принципи тестування веб-додатків детально описані в [2].

Тестування баз даних часто є дуже важливою частиною тестування інформаційних систем. Тестування бази даних вимагає всебічного знання програми, що тестується. Ключові проблеми, які виникають під час тестування баз даних, включають цілісність даних; достовірність даних (форма при введенні в бази даних); маніпулювання та оновлення даних.

Тестування розподіленої програми є досить складним завданням. Якщо програма використовує багаторівневу клієнт-серверну архітектуру, класичні підходи повинні бути реалізовані для кожного рівня.

Але в той же час вам потрібно провести інтеграційне тестування, щоб перевірити інтерфейси системи, і тестування системи, щоб перевірити зовнішні інтерфейси (зазвичай інтерфейси користувача або зовнішню систему). Для складних додатків тестування поділяється на поетапне тестування окремих компонентів, окремих модулів, окремих підсистем, окремих частин додатка; інтеграція - тестування взаємозв'язків усіх частин програми, інтерфейсу користувача, інтерфейсу системи в цілому.

Складність тестування часто зростає при створенні систем з розподіленою архітектурою - «база даних + сервер додатків + веб-інтерфейс». Для тестування цих програм потрібні додаткові витрати на «Тестування інтеграції» (тестування системного інтерфейсу) і «Тестування системи» (тестування інтерфейсу користувача). Тому актуальним залишається завдання розробки нового підходу, який полегшить цей процес ще до «події» інформаційних систем. У роботі запропоновано один із підходів, який автор називає «метод прозорої журналістики», шляхом його окремого застосування та цілих підсистем єдиної інтегрованої аналітичної інформаційної системи управління вищою освітою, яка розробляється в Тернопільському національному університеті ім. Економіку кілька років, наразі тестується.

Висновки додаються до розділу 1

У розділі 1 викладено та систематизовано основні напрями розвитку тестової техніки та суміжної дисципліни – розробки надійності програмного забезпечення. За час свого розвитку тестування еволюціонувало від невпорядкованих дій тестування, що виконуються на завершальних етапах розробки програмного забезпечення, до тестової техніки, яка охоплює всі етапи

розробки програмного забезпечення та представлена визначеними методами тестування, моделями процесу та методами оцінки його зрілості. .

Було проаналізовано різні критерії виконання тесту та відзначено їх недоліки з точки зору обмеженості ресурсів. Це підтверджує актуальність проблеми з точки зору розробки критеріїв і методів, які б враховували ресурсні обмеження, в основному за витратами і часом, а також ризики відмов.

РОЗДІЛ 2
РОЗРОБКА МЕТОДІВ ТА ІНСТРУМЕНТІВ ДЛЯ ОРГАНІЗАЦІЇ ПРОЦЕСУ ТЕСТУВАННЯ В ВЕБ-ОРІЄНТОВАНИХ СИСТЕМАХ

2.1. Інтеграційне тестування розподіленої системи

Для демонстрації запропонованого підходу на практиці ми використовуємо реальну інформаційну систему з наступною розподіленою архітектурою (рис. 2.1): клієнт отримує доступ до програми через веб-браузер; сервер додатків обробляє запит, при необхідності вставляє отримані за запитом дані від сервера баз даних, створює вихідний файл HTML; створена сторінка надсилається клієнту.

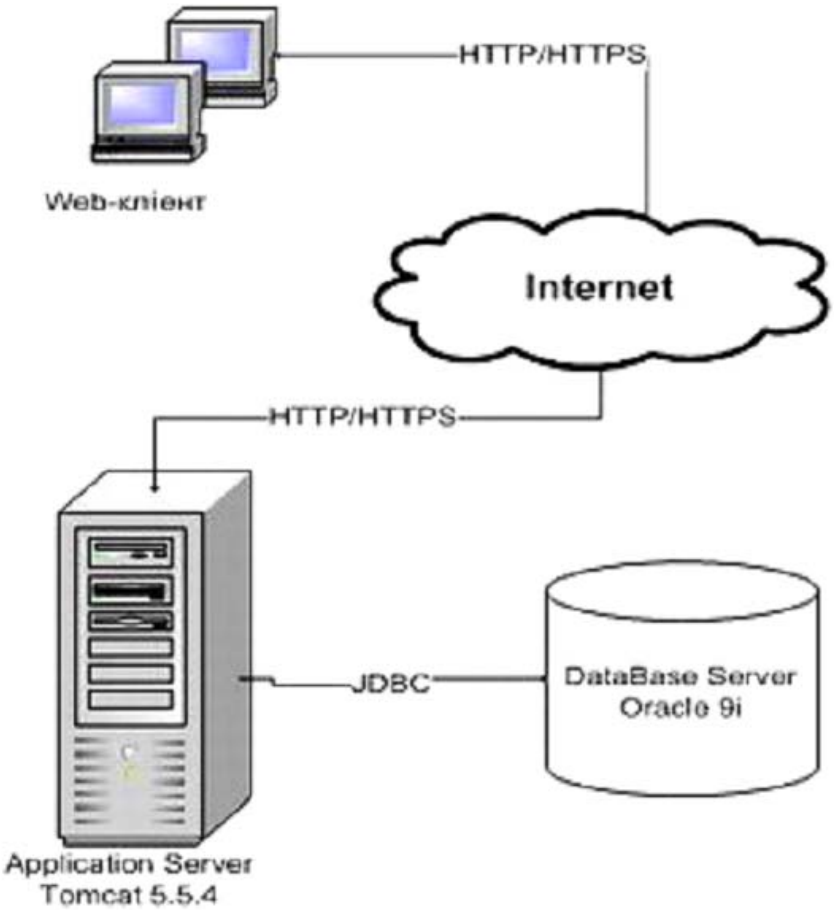

Рис. 2. 1 . Архітектура розподіленої системи

На сервері додатків обробка даних виконується пакетом Kemsu WEB, структура якого показана на рисунку 2.2. Основним носієм інформації є XML-файл шаблону майбутньої сторінки з даними. Шаблон містить спеціальні конструкції, призначені для розширення можливостей обробки даних мови XML: виклик процедур або виконання запитів для отримання даних від сервера бази даних; управління логічними інструкціями; обробка локальних змінних і змінних сеансу (глобальних для цього сеансу користувача).

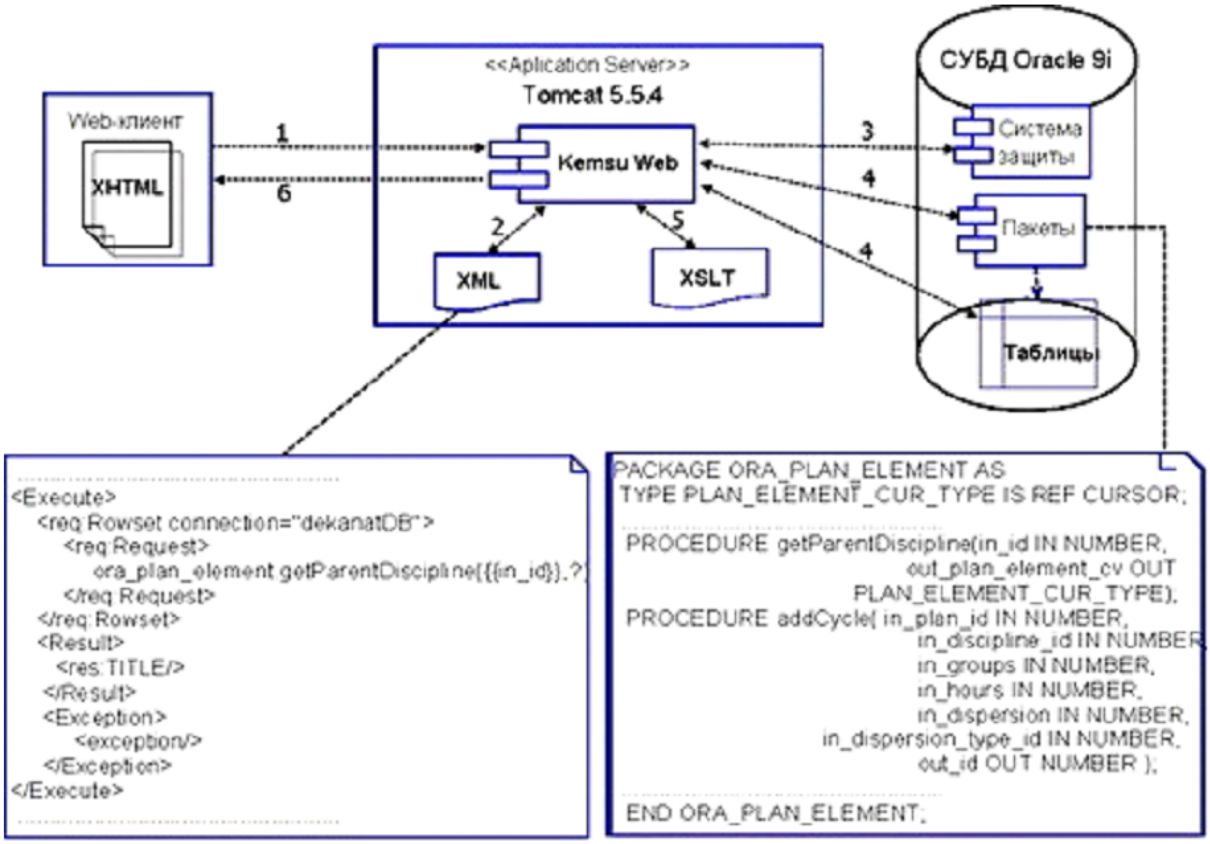

Рис. 2.2. Архітектура пакету Kemsu WEB

Іншими словами, сервер додатків розділяє рівні зберігання даних, обробки даних і створення інтерфейсу користувача (рис. 2.3).

Рис. 2.3. Рівні застосування

Процес тестування такого додатка є складним завданням. Для її вирішення необхідно було поетапно тестувати окремі компоненти програми, інтерфейс користувача, коректність роботи запитів на дані тощо. Необхідно накопичувати статистику про виконання тестів і стан запиту, в порядок визначення структури тестової системи. Крім того, під час виконання тестових випадків бажано мати можливість швидко змінювати структуру програми та визначати гілку графа структури програми, яку необхідно перевірити. Набір значень вхідних параметрів, заданих через форми веб-інтерфейсу, також повинен накопичуватися разом з відповідними відповідями системи.

Техніка «прозорих журналів» призначена для проведення тестових дій, які будуть пояснені нижче на прикладі тестування лише невеликої частини авторизованої компонентної системи на конференц-сервері ACIT.

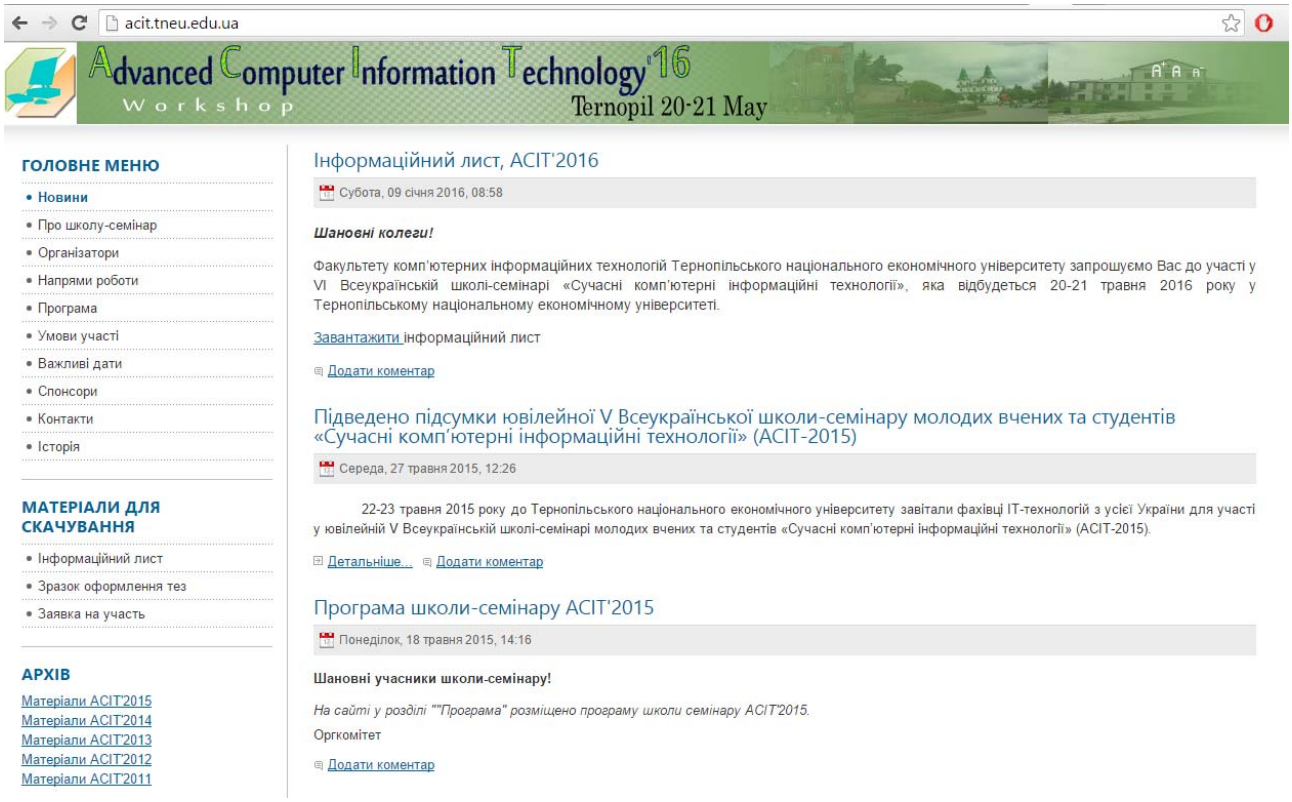

Рис. 2.4. Приклад веб- додатку для демонстрації методології тестування

2.5 показано фрагмент вихідного коду інтерфейсу користувача, реалізованого у відповідному файлі XML.

```
<center><table style="font-size: 11pt">
[1]  <res:If name="$$PersonID$$" value="" op="eq">
     <res:Then>
          <tr>
          <td>  Форм.. </td>
[2]       <form action="confi{{conferenceAlias}}/reg/ins_person.htm" method="post">
          <input type="hidden" name="in_lang" value="U"/>
          <td align="center" colspan="2"><input type="submit" class="btn" value=</td>
          </form>
          </tr>
          <tr>
          <td></td>
[3]       <form action="confi{{conferenceAlias}}/reg/chk.htm" method="post">
          <td align="right" style="font-size: 10pt">
                    Login: <input type="text" name="in_login" value="" size="10"/><br/>
                    Пароль: <input type="password" name="in_passwd" size="10"/><br/><br/>
                    <input type="submit" class="btn" value="Ві йти"/>
          </td>
          </form>
          </tr>
</res:Then>
<res:Else>
          <tr>
[4]       <th align="left">Авторизован </th>
          <td align="right" style="font-size: 10pt" colspan="2">
                    <i>##LAST_NAME## ##FIRST_NAME## ##MIDDLE_NAME##</i>
                    <br/>##ORGANISATION##<br/>##CITY##<br/>##COUNTRY##
          </td>
</res:Else>
 </res:If>
</table></center>
```

Рис. 2.5 _ _ Фрагмент вихідного коду

Оскільки цей файл має текстовий формат і завжди доступний для перегляду/редагування (в рамках доступних привілеїв системи безпеки), він дає тестувальнику кілька переваг, коли веб-обізнаний і впорядкований граф структури програми.

Для демонстрації запропонованої методології ми використовуємо лише частину форми (рисунок 2.4), що визначає поведінку системи у двох випадках:

- якщо користувач відвідав сайт конференції вперше, він повинен ввести свої реєстраційні дані (натиснути кнопку «заповнити»);

- якщо користувач вже має персональні дані в цій системі, йому необхідно пройти авторизований вхід для подальшої роботи (заповнити поля у формі «Логін» і «пароль», натиснути кнопку «вхід»).

2.2. Метод «прозорих журналів» для організації процесу тестування веб - орієнтованих інформаційних систем

Суть методу полягає в наступному. У вихідний файл XML тестер спеціальним чином додає мітки. Причому при виконанні цього коду мітки разом з фрагментами результуючого коду і значеннями відповідних змінних поміщаються в спеціальну таблицю - журнал активності. Формат запису в простому випадку наведено в таблиці 2.1.

Таблиця 2.1

Формат запису журналу активності

Назва модуля (компонента).	Мітка вузла	Позначте наступний вузол	Дія / передбачення	Прогностичне значення	Повернене значення	трафік

Кожен запис є елементом графа структури тестованого додатку. Набір записів повністю представляє таблицю на графіку. Фрагмент графа структури

наведено на рисунку 2.6, а відповідний набір записів у журналі активності – у таблиці 2.2.

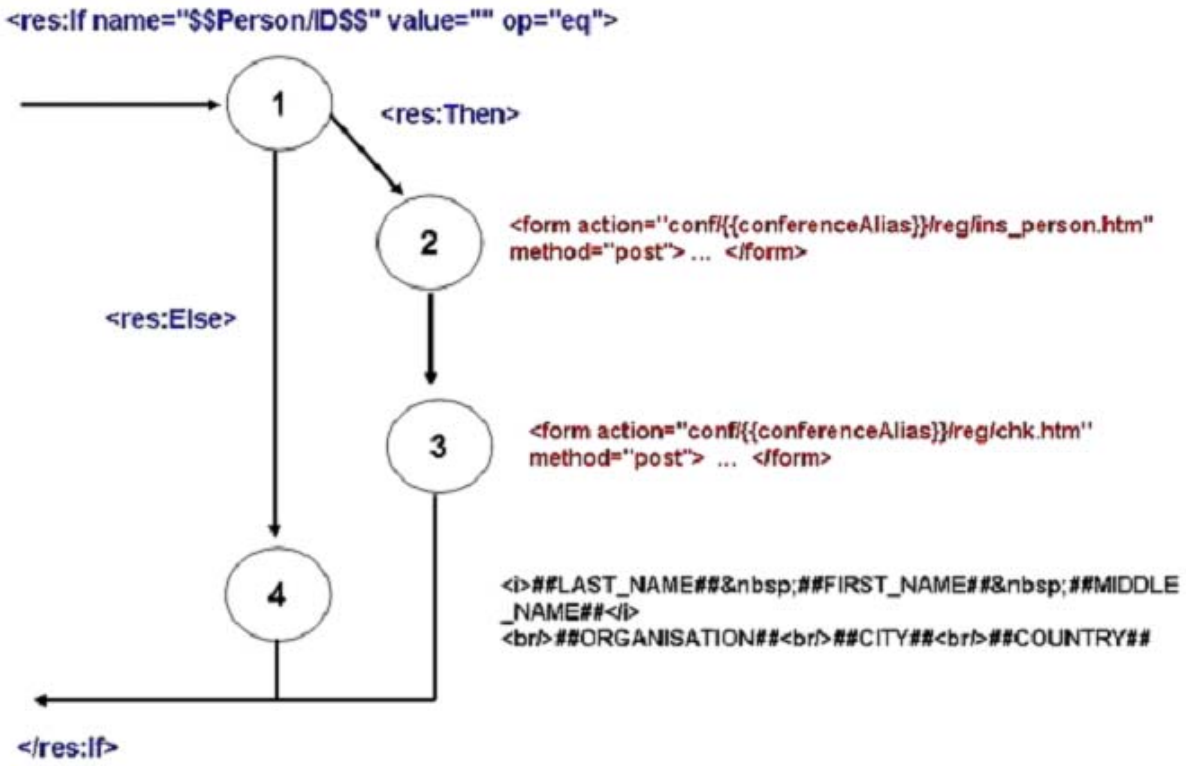

Рис. 2.6. Фрагмент графа структури програми

У рядку таблиці з номером 1 перевіряється статус авторизації користувача (вузол з номером 1 на малюнку 1). Якщо значення змінної $$Person/ID$$ не визначено, то користувач не зареєстрований в системі і він повинен заповнити свої дані за допомогою спеціальної форми (вузол номер 4 на малюнку 2.6). .

Якщо користувач почав процес реєстрації в системі (вузол номер 4 на малюнку 2.6) за допомогою спеціальної форми, то її параметри можна розмістити в колонці 4 таблиці 2.2 (змінні #ПРІЗВИЩЕ ##, ## ІМ'Я # #, ## MIDDLE_NAME ## , ## ORGANIZATION ##, ## CITY##, ##COUNTRY ##) і може використовуватися для детального тестування процесу реєстрації.

Таблиця 2.2

Приклад формату записів журналу активності

Назва модуля (компонента).	Мітка вузла	Позначте наступний вузол	Дія/ пророцтво	Прогностичне значення	Повернене значення	трафік
http :// acit . tnew . освіта . u / admin /	[1]	[2]	Якщо назва = "$$Особа/ID$$" value="" op="eq"	правда		Людська цінність/ ID = нуль
http :// acit . tnew . освіта . u / admin /	[2]	[3]	conf/{{conferenceAlias}}/reg/ins_person.htm	Значення параметрів		Перехід на форму управління
http :// acit . tnew . освіта . u / admin /	[1]	[4]	##ПРІЗВИЩЕ## ##ІМ'Я## ##ПРІЗВИЩЕ# ##ОРГАНІЗАЦІЯ#	Значення змінних	Звідки вони повернулися?	Вивід у файл XML

Після завершення необхідних тестів тестеру доступна структура частин системного модуля (компонента), набір значень для переданих/отриманих змінних, набір системних повідомлень (розміщуються в останньому полі) у разі виникнення проблем. На основі цих даних можна проаналізувати охоплення структури програми. За допомогою аналізу покриття можна оцінити ефективність тесту (за допомогою спеціалізованих інструментів тестування) і подивитися інструкції виконуваного коду. Журнал активності зберігається в спеціальній схемі бази даних, де накопичуються необхідні статистичні дані.

Найпростіший алгоритм методу «прозорих журналів» представлено на рисунку 2.7.

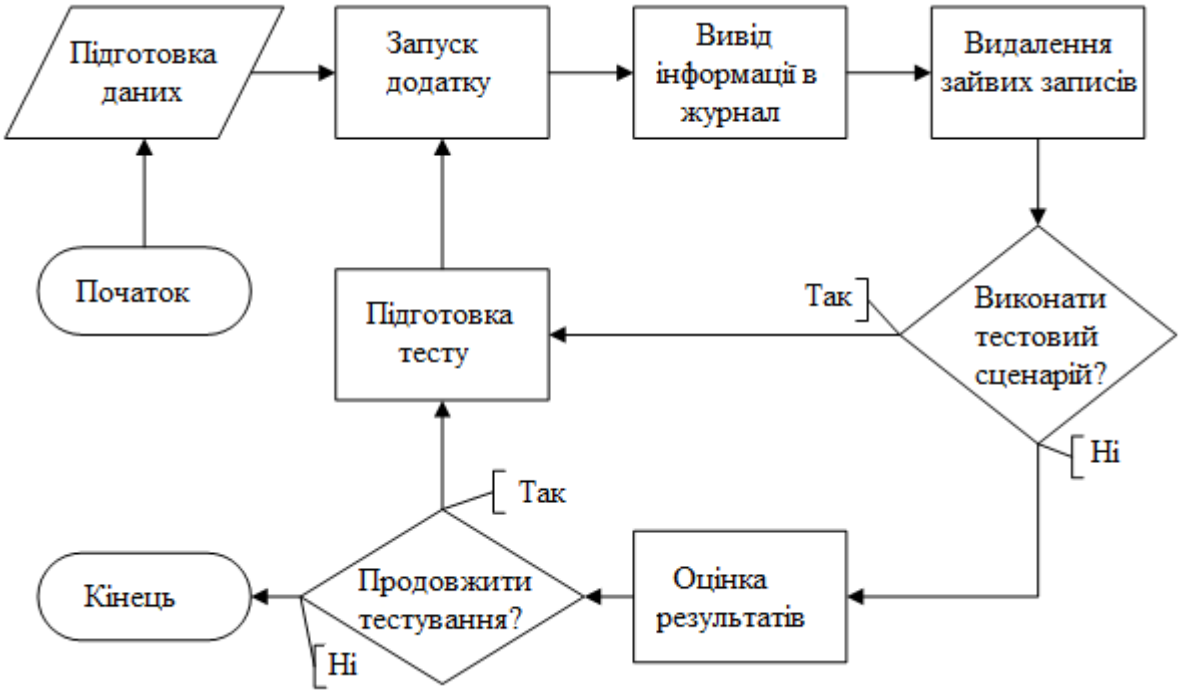

Рис. 2.7. Найпростіший алгоритм методу «прозорий журнал».

2.3. Визначте основний процес тестування

У розробці програмного забезпечення процес розглядається як упорядкована сукупність елементів - дій (завдань), входів (вхідних даних для виконання процесу) і виходів (результатів процесу) [2]. Основна мета базового процесу тестування полягає у визначенні цих елементів, критеріїв для початку та завершення завдань, ролей та обов’язків під час виконання тестових завдань на літаку.

Під час побудови цього процесу були досліджені всі процеси житлового комплексу, визначені ДСТУ 3918, та визначені тестові завдання, розподілені між цими процесами. Потім усі ці завдання були об’єднані в один безперервний процес тестування.

Процес представлений серією завдань для підготовки , проведення та оцінки результатів тестування, які розділені на 10 кроків процесу.

Модель процесу тестування наведена на рис. 2.8. На кожному етапі *підготовки* робочі продукти відповідного процесу розробки (вхідні дані для цього етапу процесу тестування) аналізуються для визначення цілей, завдань, сценаріїв і ресурсів тестування, адекватних рівню тестування. Результати етапів підготовки до тестування фіксуються в планах тестування.

На кожному етапі *виконання* результати випробувань записуються та порівнюються з очікуваними результатами.

аналізуються для визначення поточного стану програмного забезпечення та прийняття рішення про адекватність тестування на цьому рівні.

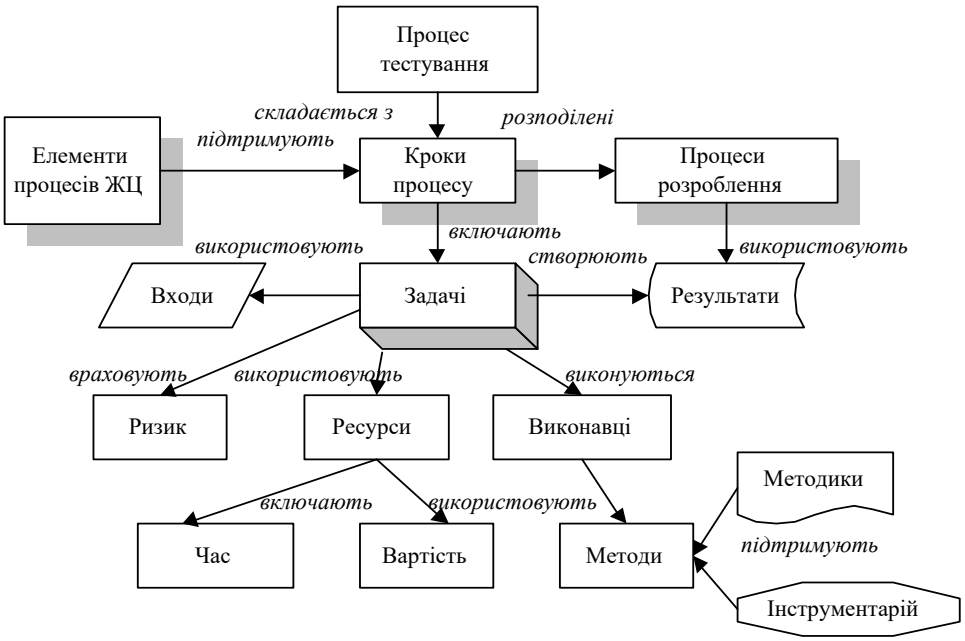

Рис. 2.8. Модель базового процесу тестування

завдань на кожному кроці процесу. Розподіл етапів і тестових завдань за процесами розробки, визначеними в ДСТУ 3918, схематично наведено на рисунку. 2.9 . Кроки процесу та окремі завдання можуть виконуватися циклічно для різних об’єктів тестування та рівнів.

Модель використовується для визначення оптимального часу тестування для визначення критерію завершення тестування програмного забезпечення (крок 7).

Методика оцінки ризику невдач входить до складу завдань для підготовки тесту кожного рівня (кроки 2-5) та оцінки результатів тестування окремих модулів.

Визначення базового процесу включає : розподіл обов'язків між учасниками процесу, вимоги до професійної підготовки виконавців процесу, стандарти подання документів, метрики процесу, застосовні методи розв'язання тестових завдань, критерії для початку та завершення завдань, завершення та переходу до наступного. крок. процесу. Для документування процесу тестування розроблено спеціальні шаблони документів, що стосуються кожного етапу процесу тестування.

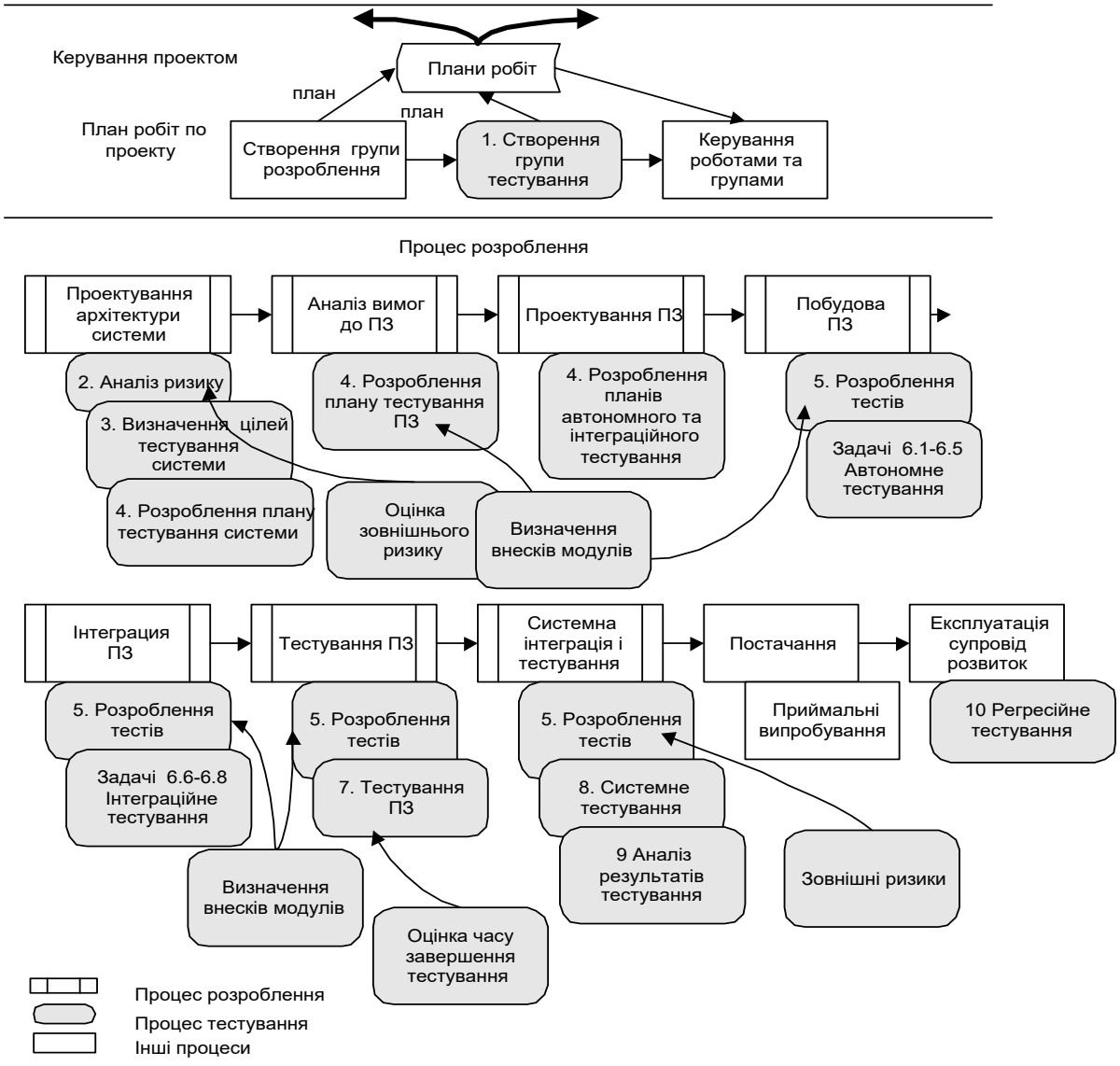

Рис. 2.9. Структура базового тестового процесу з поділом за процесами ЖЦ

Перелік кроків і завдань процесу наведено в таблиці 2.3.

Таблиця 2.3

Склад завдань основного процесу тестування

Етап процесу	Завдання процесу
1. Створити тестову групу	1.1.Визначити учасників процесу тестування
	1.2. Розподіліть обов'язки та сформулюйте план роботи тестової групи
2. Аналіз ризиків	2.1. Виявлення ризиків
	2.2. Розстановка ризиків
	2.3. Розподілення ресурсів
3. Визначте цілі тесту	3.1. Визначте цілі тесту
	3.2. Визначення критеріїв складання тестів
	3.3. Визначення цілей тесту через оцінку ризику
4. Розробити плани тестування	4.1. Розробіть план тестування PS
	4.2. Розробіть план тестування програмного забезпечення
	4.3. Розробіть план тестування інтеграції
	4.4. Розробіть автономний план тестування
	4.5. Розробіть план тестування регресії
5. Розробити тести	5.1. Проектування та розробка тестів
	5.2. Підготовка тестових даних
	5.3. Підготуйте протоколи тестування
	5.4. Перевірка тестових документів
6. Автономне тестування та інтеграція	6.1. Автономне тестування
	6.2. Повторне випробування після усунення дефектів
	6.3. Аналіз результатів автономного тестування
	6.4. Інтеграційне тестування
	6.5. Повторне випробування після усунення дефектів
	6.6. Аналіз результатів інтеграційного тесту
7. Тестування ПЗ	7.1. Затвердження тестового середовища
	7.2. Затвердження тестового ресурсу
	7.3. Тестування програмного забезпечення
	7.4. Повторне випробування після усунення дефектів
	7.5. Аналіз результатів тестування
8. Тестування системи	8.1. Затвердження тестового середовища
	8.2. Затвердження тестового ресурсу
	8.3. Тестування системи
	8.4. Повторне випробування після усунення дефектів
	8.5. Аналіз результатів тестування
	8.6. Перевірте установку
9. Аналіз результатів тестування та	9.1. Аналіз даних результатів тестування
	9.2. Готувати рішення та рекомендації за результатами тестування

Етап процесу	Завдання процесу
підготовка звіту	9.3. Підготовка підсумкового звіту за результатами тестування
	9.4. Розгляд рішень та звіт
10. Регресійне тестування	10.1. Проведення регресійного тестування
	10.2. Аналіз результатів регресійного тесту

Розподіл обов'язків між виконавцями процесу наведено в таблиці 2.4.

Таблиця 2.4

Розподілити обов'язки між виконавцями процесу тестування

Крок у процесі тестування	Учасники процесу
Створіть тестову групу	Керівник проекту
Аналіз ризиків	Керівник проекту, керівник групи тестування, аналітик проекту, група забезпечення якості , користувачі
Визначте цілі тесту	Керівник групи тестування, тестувальники, група забезпечення якості , користувачі
Розробити тестові плани	Керівник залікової групи, випробувачі
Розробка тестів	Розробники, тестери
Автономне та інтеграційне тестування	Розробники, тестери
Тестування програмного забезпечення	Тестери
Тестування системи	Тестери
Аналіз результатів тестування	Керівник проекту, керівник групи тест, тестери, група контролю якості
Регресійне тестування	Тестери

У таблиці 2.5 наведені вимоги до професійної підготовки членів тестової групи.

Таблиця 2. 5

Вимоги до професійної підготовки випробувальної групи

Роль	Вимоги до професійної підготовки членів тестової групи
Лідер групи	• Значний досвід тестування. • Технічні знання та навички в предметній галузі (програмний продукт). • Знання компонентів інфраструктури (апаратне та програмне забезпечення системи). • Знання вимог і стандартів тестування.
Тестер	• Технічні знання та навички в області продуктів/технологій, що використовуються для створення програмного забезпечення. • Знання компонентів інфраструктури. • Знання вимог і стандартів тестування. • Право власності на засоби тестування. • Досвід юзабіліті тестування. • Знання стандартів/вимог юзабіліті.

IEEE Std 829:1998 для документування процесу випробувань, зокрема розробки планів випробувань літаків і остаточних звітів на основі результатів випробувань. Стандарт документації тестування програмного забезпечення [15]. Також можна використовувати стандарти ДСТУ (для перевірки) [16, 17].

Для опису проміжних робочих продуктів процесу розроблено шаблони документів процесу:

- протокол розподілу обов'язків щодо виконання тестових завдань;
- план роботи тестової групи;
- реєструвати ризики;
- опис процедур, наборів тестів і тестових випадків;
- журнали випробувань;
- паспорти випробувального обладнання;
- повідомлення про проблеми.

Взаємозв'язок документів у процесі тестування показано на рисунку 2.10.

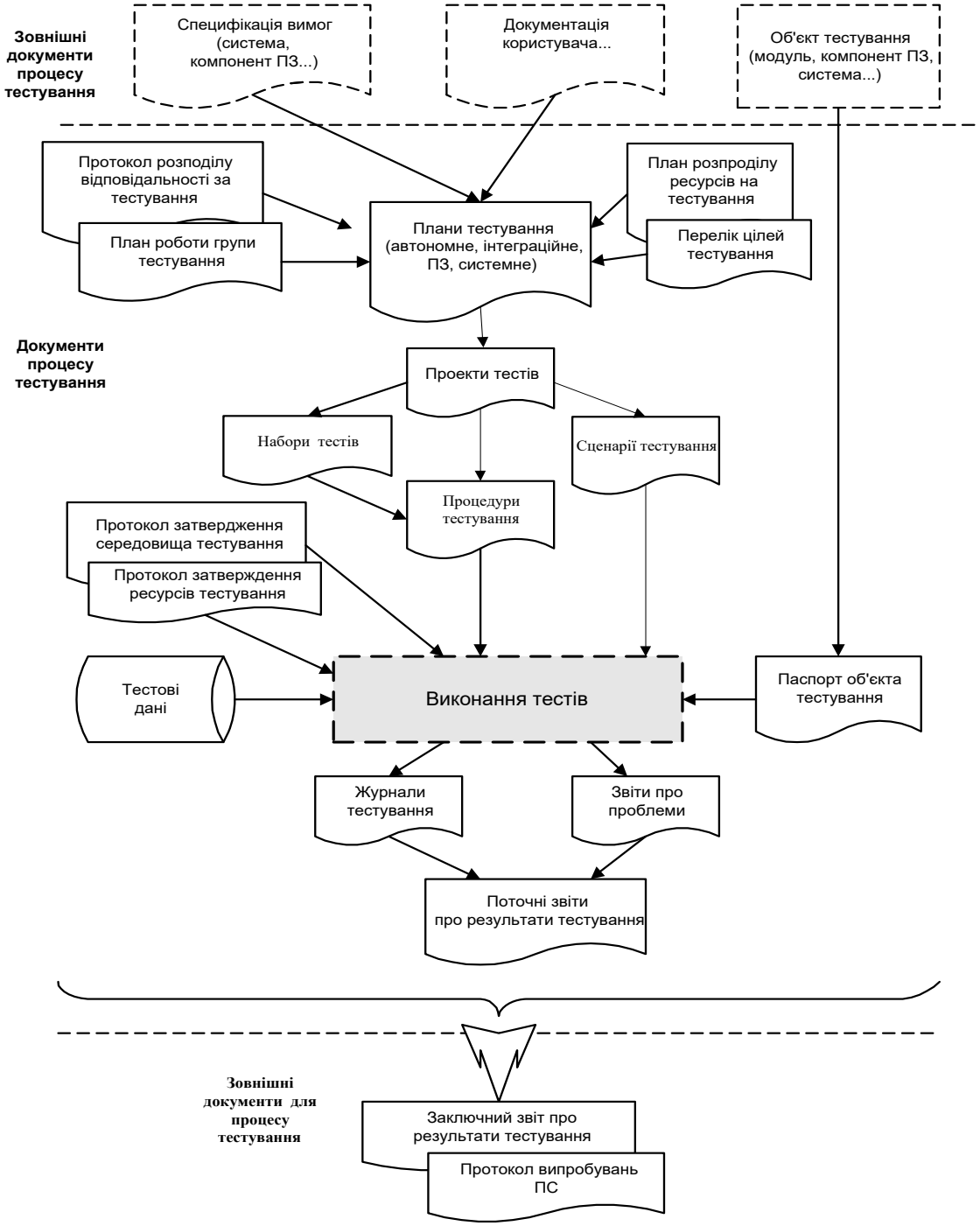

Рис. 2. 10. Взаємозв'язок документів процесу тестування

Для аналізу поточної продуктивності процесу тестування були встановлені такі показники:

1. Показники дефектів:

- кількість дефектів (виявлених, усунених, не усунених);

- типи несправностей і ступінь тяжкості.

2. Показники ресурсів:

- час тестування в календарних одиницях часу та час виконання;

- вартість тесту;

- час і вартість усунення дефекту.

Вимірювання проводяться в процесі тестування, а їх результати відображаються в безперервних звітах. Структура процесу тестування включає три види проміжних (щотижневих) звітів про поточний стан тесту.

Висновки розділу 2

У цьому розділі проведено теоретичні дослідження щодо основних методів тестування веб- систем. Метод «прозорих журналів» рекомендується для тестування веб- систем . Описано процедуру формулювання основного тестового процесу.

Розділ 3
РЕАЛІЗАЦІЯ ЗАПРОПОНОВАНИХ МЕТОДІВ У ЗАГАЛЬНІЙ СИСТЕМІ ТЕСТУВАННЯ ВЕБ - ОРІЄНТОВАНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

3.1. Аспекти програмної реалізації модуля в системі тестування веб-додатків

Java для реалізації модуля. Java це скомпільована мова програмування, тобто програми, які вона створює, перетворюються у виконуваний файл за допомогою компілятора. У такому процесі трансформації є два етапи. По-перше, програмний код на мові Java за допомогою програми-транслятора він перетворюється в так званий байт-код, який можна перетворити в машинний код за допомогою віртуальної машини Java – Ява віртуальний машина (JVM) (Малюнок 3.1). Це дозволяє пізніше запускати програму, написану мовою програмування Java на комп’ютерах під керуванням різних операційних систем. Єдина вимога для запуску програми – встановити на комп’ютері відповідну віртуальну машину Java. Таким чином досягається максимальна мобільність створюваних програм і їх незалежність від платформи виконання.

Мова програмування Java заснований на принципах об'єктно-орієнтованого програмування. Основа будь-якої мовної програми Java це клас Кожен клас містить властивості та методи. Властивості — це характеристики об’єкта, тоді як методи — це процедури та функції, які застосовують дії до об’єкта та його властивостей.

Кожен створений клас представляє тип даних. Для використання в програмі створюються так звані «об'єкти» або екземпляри класу. При створенні кожного нового об’єкта кожному екземпляру класу надаються властивості, які він повинен мати, а також методи роботи з ними.

Наприклад, якщо створюється програма для роботи зі списком тестів, то створюється клас, на основі якого формується набір об’єктів, кожен з яких

зберігає інформацію про певний тест. Властивості кожного екземпляра класу можуть бути: ім'я, дата, ідентифікатор, тип тесту та інші. Методи цього класу — це процедури та функції, призначені для створення нового тестового запису, видалення тесту зі списку, отримання шифру, зміни тесту тощо.

Завантажте програму взаємодії мовою Java з користувачем і зовнішнім середовищем будується на обробці подій. Подіями можуть бути: натискання користувачем кнопки, виклик пункту меню, зміна візуальної складової на екранній формі та ін. Коли відбувається подія, автоматично викликається так званий обробник події, який, у свою чергу, запускає вказану підпрограму на виконання. Основні методи класу програми діють як підпрограми обробки подій.

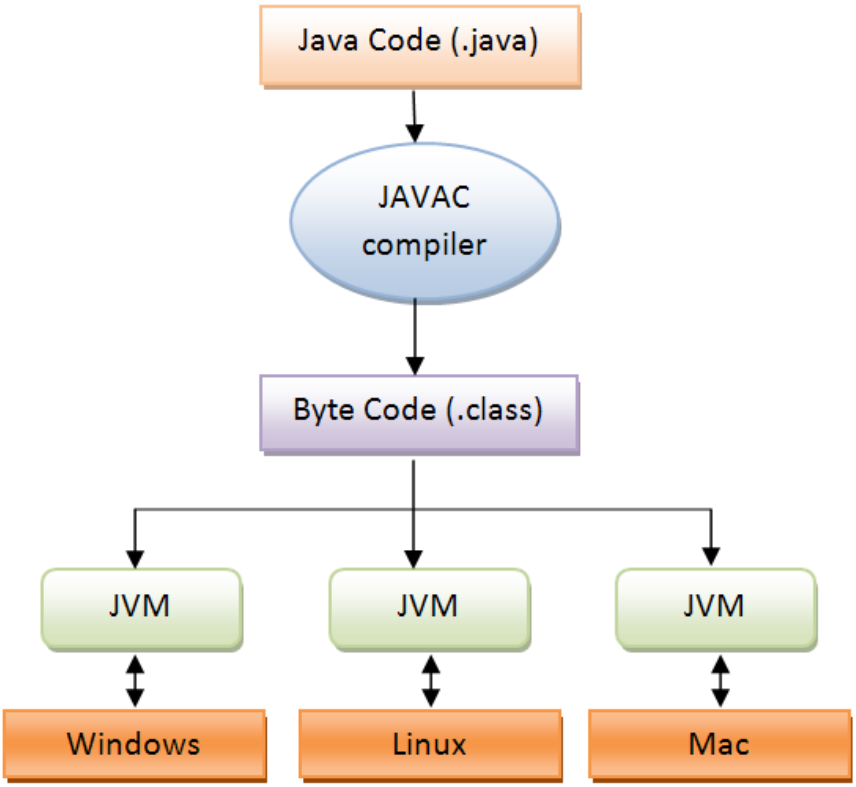

Рис. 3.1. Схема реалізації java програм

JSP (Java Server Pages) — це технологія, яка дозволяє веб-розробникам динамічно створювати HTML , XML та інші веб-сторінки. Робота над JSP почалася в 1997 році. Пізніше JSP було включено в Java EE , програмну

платформу для програмування веб-додатків. Технологія дозволяє вставляти код Java в статичний вміст сторінки. Бібліотеки тегів JSP також можна використовувати для вставки їх у сторінки JSP. Компілятор JSP компілює сторінки в сервлети , які є класами Java, і виконує їх на сервері. Розробник також може писати сервлети без використання сторінок JSP. Ці технології доповнюють одна одну.

JSP є однією з високопродуктивних технологій, оскільки весь код сторінки перетворюється на код сервлету Java за допомогою компілятора сторінок JSP (наприклад, Jasper), а потім компілюється у допоміжний код віртуальної машини Java (JVM) . Нижче наведено приклад сторінки jsp

```
<? xml версія = "1.0" кодування = "UTF-8" ?>
< jsp : корінь xmlns : jsp = "http://java.sun.com/JSP/Page" версія = "2.0" >
    < jsp : директива . сторінки contentType = "application/xhtml+xml;
charset=UTF-8" />
    < jsp : вихід doctype - root - element = "html" doctype - public = "-
//W3C//DTD XHTML 1.1//EN"
        doctype - система = "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd" omit
- xml - declaration = "true" />
    < html xmlns = "http://www.w3.org/1999/xhtml" >
    < голова >
    < мета http - equiv = "Тип вмісту" content = "text/html; charset=UTF-8" />
    < title > Методика тестування «Прозора журналістика» </ title >
    </ head >
    < тіло >
    < h1 > Тест </ h1 >
    < p > Журнал тестування </ p >
    < jsp : скриптлет >
    поза . print ( Calendar . getInstance ( запит . getLocale ()). getFirst Test
() == Тест . НОМЕР ?
        «Успішний старт тесту» :
        «Помилка тестового запуску» );
    </ jsp : скриптлет >
    </ body >
    </ html >
</ jsp : корінь >
```

3.2. Опис реалізації модуля в системі тестування веб- додатків

Робота модуля в системі починається на етапі виконання будь-якого тесту, тобто коли виконується метод doGet () сервлета NThread . Модуль запускається при виклику сторінки Test JSP , на якій відображається вся інформація про проходження тесту. Однак перед його наданням встановлюються певні значення атрибутів сеансу, наприклад:

1. StratTime — це календар, який містить деталі початку тесту.
2. EndTime – об’єкт Calendar , що містить дані про закінчення тесту.
3. Додаткові атрибути для відображення графіків

Після завершення тесту керування переходить на початкову сторінку тесту . jsp (рис. 3.2.), де обробляються параметри сеансу та виводиться на екран сторінка з текстовими результатами.

Паралельний виклик події MyChart повертає графічні результати . діаграма . ВЕБ - сервер перехоплює виклик цієї події та відповідно до вмісту веб- файлу . xml вказує , що необхідно виконати сервлет ChartServlet , який відповідає за генерацію графічного зображення результатів тесту.

ChartServlet працює за таким алгоритмом:

1. Визначення імені класу графа, його подальша генерація.
2. Виклик додаткового класу ChartEngine , який призначений для аналізу xml -файлу конфігурацій графа та отримання його параметрів із цього файлу з наявною назвою графіка.

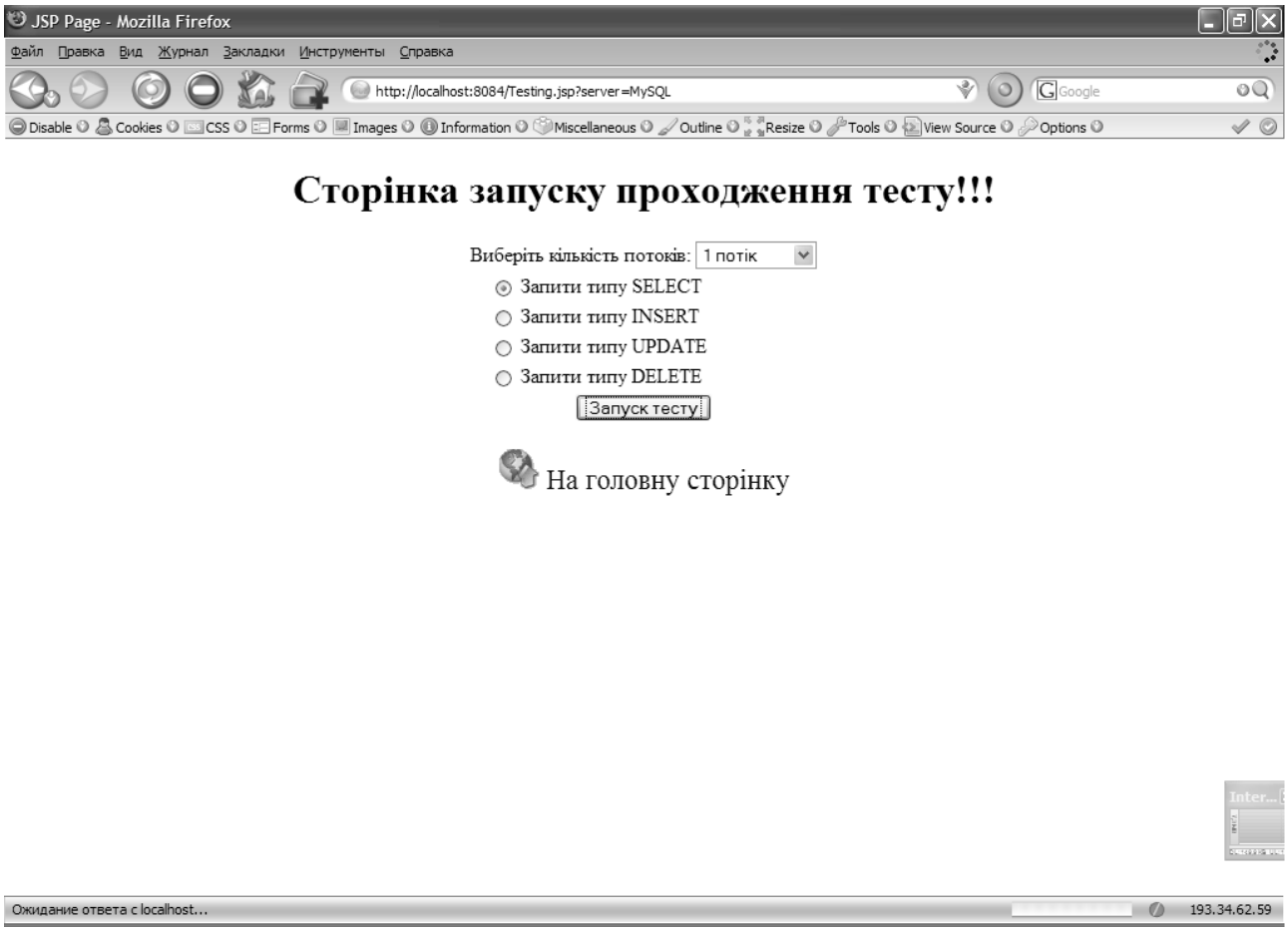

Рис. 3. 2 Сторінка початкового тестового запуску

3. Викличте реалізацію класу ChartProducer , який призначений для створення певного графіка на основі наявних даних.

4. Збережіть отримане графічне зображення у тимчасовій директорії.

5. Перенесіть графіку на тестову сторінку як малюнок для відображення.

згенерованого зображення сервлет Test_jsp . клас він показує його внизу сторінки під текстовими деталями результатів тесту.

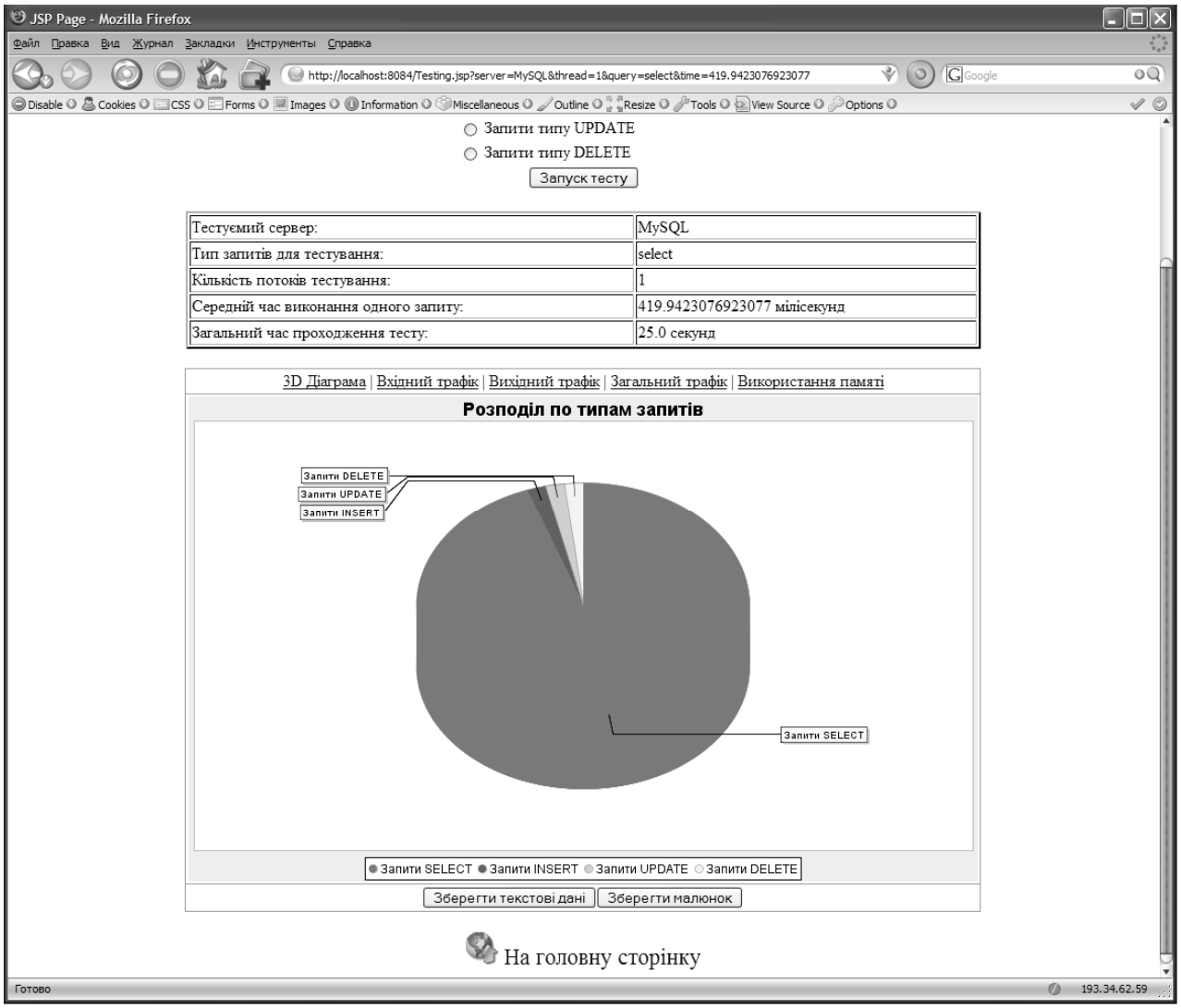

Рис. 3. 3 Графічний результат розподілу за типами додатків.

Графічні результати тестування можна відобразити у вигляді графіка для таких елементів:

1. Розподіл за типами запитів (показує співвідношення кількості різних типів запитів для конкретного тесту).
2. Розподілити навантаження на мережу через вхідний трафік.
3. Розподіл навантаження на мережу за вихідним трафіком.
4. Розподіл навантаження мережі за загальним трафіком (включаючи вхідний і вихідний трафік).
5. Виділення та використання оперативної пам'яті віртуальною машиною Java .

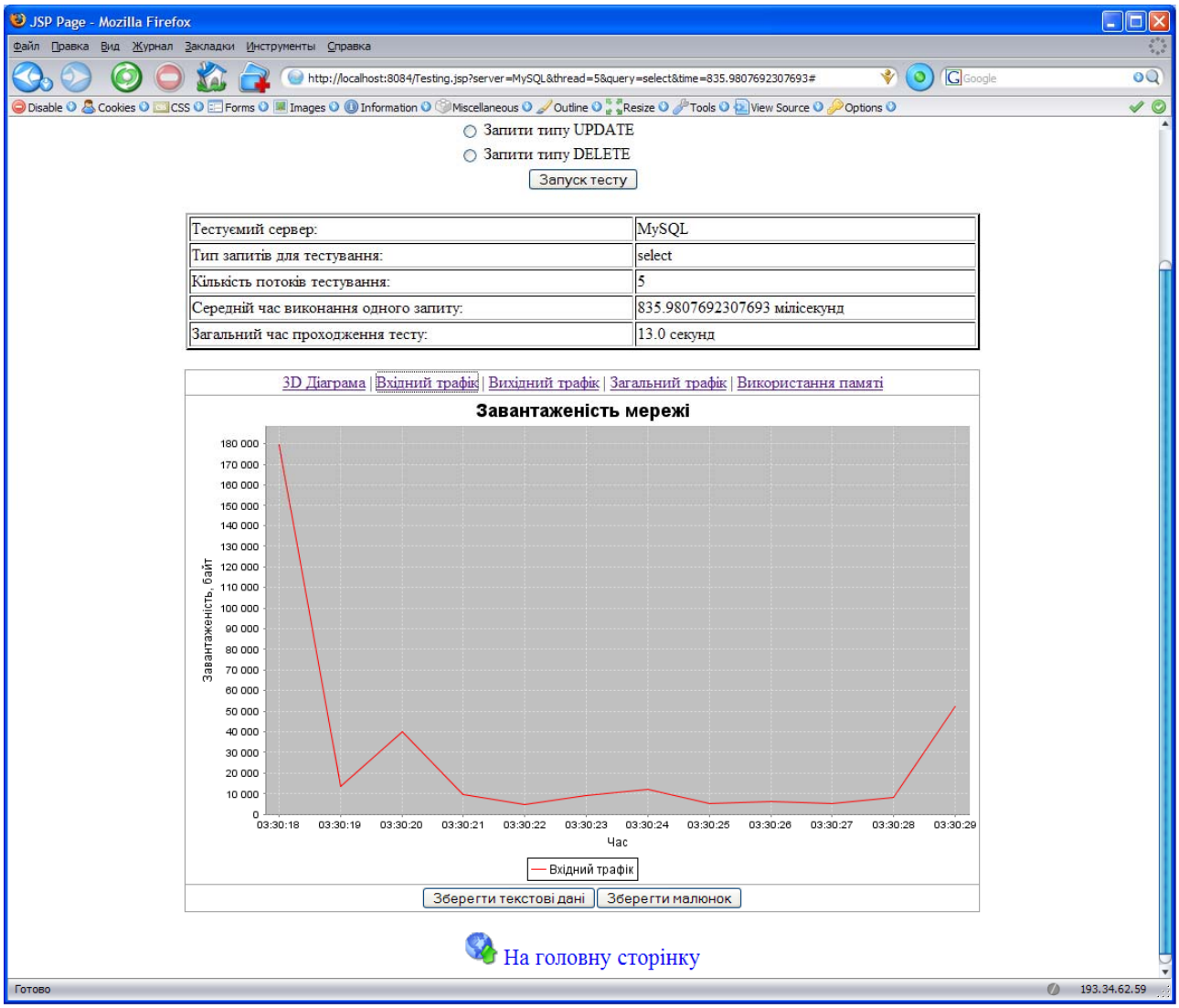

Рис. 3. 4 Відобразити графік вхідного трафіку.

На графіку показано стан завантаження програми та вхідний трафік, починаючи з моменту натискання кнопки «Почати тест» і закінчуючи в кінці тесту. На діаграмі показана залежність переданої інформації від часу. При цьому, якщо час проходження тесту досить великий, діаграма автоматично масштабується по обох осях.

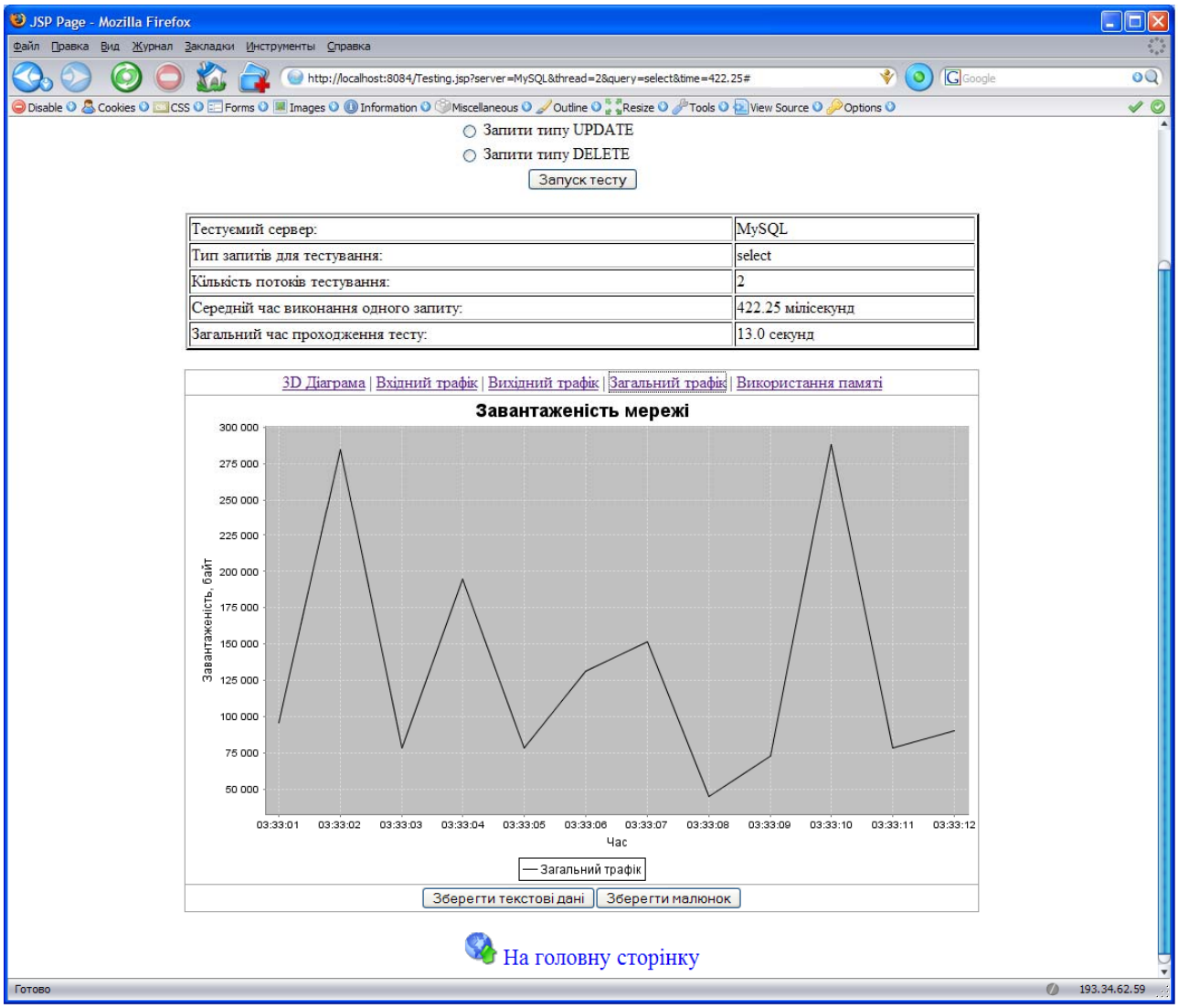

Рис. 3.5 _ _ Графік загального навантаження мережі під час тесту

Останнім типом графіка, реалізованого в цьому модулі, є графік використання оперативної пам'яті віртуальною машиною Java . Цей графік відрізняється тим, що він динамічний і змінюється, щоб показати зміни під час тесту. Найголовніше, щоб він був показаний користувачеві в окремому вікні.

Як ви можете бачити, на графіку нижній рядок показує, скільки оперативної пам’яті зараз використовується віртуальною машиною Java , а верхній рядок показує максимальний обсяг доступної для неї пам’яті. При цьому необхідно пам'ятати, що в пам'ять також входить файл підкачки, який знаходиться на жорсткому диску. Тому іноді можуть виникати випадки, коли

обсяг використовуваної оперативної пам’яті може бути більшим, ніж фактично доступний на комп’ютері, на якому розташований веб -сервер .

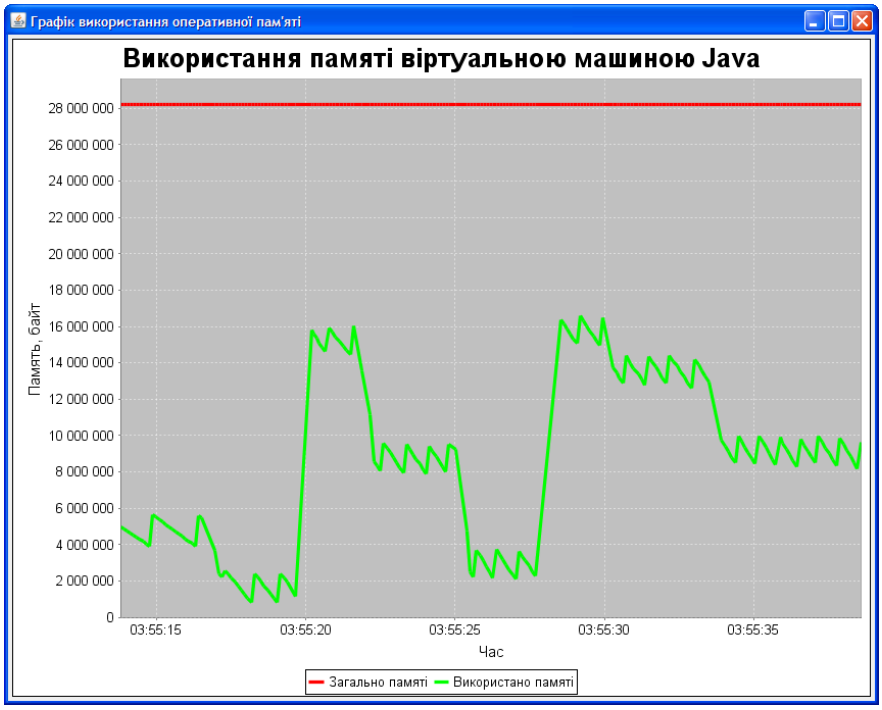

Рис. 3. 6 Вікно використання оперативної пам'яті віртуальною машиною Java .

Щоб контролювати вигляд графіків, їх можна змінювати за допомогою контекстного меню, в якому вибирається пункт «Параметр». У вікні, що з’явилося, можна задати всі необхідні параметри розкладу:

1. Тип і розмір шрифтів.
2. Наявність або відсутність координатних осей.
3. Наявність або відсутність проміжних ліній.
4. Вибір кольорів для ліній, графіки, фону тощо.

Модуль розробки дозволяє зберігати дані у вигляді зображень, які будуть відображатися в браузері, або як текстові дані в текстовому файлі.

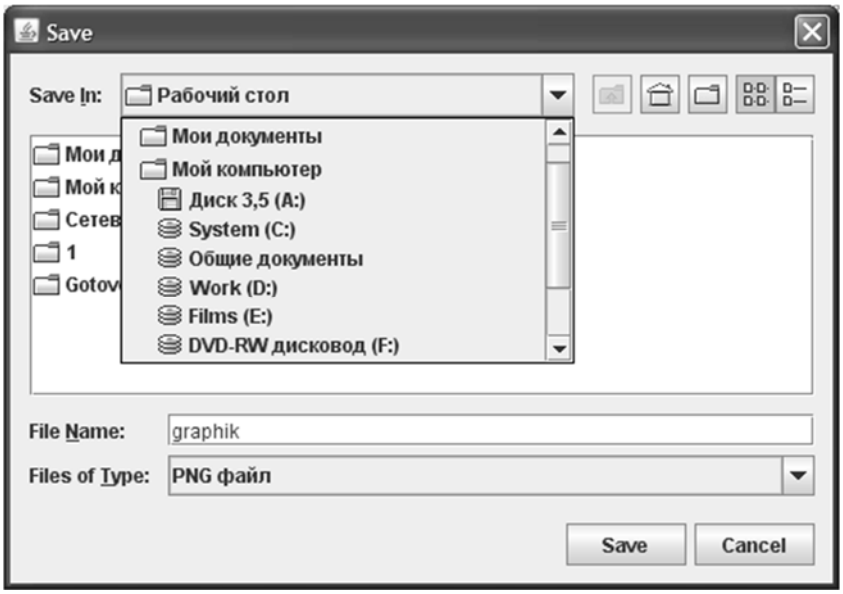

Рис. 3. 7 Діалогове вікно для збереження графіка у форматі PNG.

Недоліком процесу збереження графіки є те, що її можна зберегти лише як зображення у форматі png. Щоб отримати інші типи малюнків, необхідно використовувати зовнішні редактори для їх зміни.

Для того, щоб система працювала, розроблено певний набір класів, які реалізують процеси встановлення початкових параметрів, виконання тесту, генерації серії графів даних і самого графа, збереження та передачі файлів графів у браузері, зміна аспектів побудованих діаграм. . Класи розробки модулів включають:

1. Оновлений тестовий клас.
2. Оновлений клас NThread.
3. Класи різних типів діаграм (MyChart , MyChart 2, MyChart 3, MyChart 4, MyChart 5, MemoryUsageDemo).
4. Клас ChartServlet .
5. Chart Engine, Chart Reporter, Chart Producer .
6. PathTag .
7. ParseData, статистичні дані .
8. Додаткові заняття зі зміни зовнішнього вигляду графіків.
9. Файли конфігурації та файл журналу.

Клас Test призначений для вибору та відображення основних параметрів тестів, а також виведення результатів тестів. Для своєї роботи він використовує всі описані нижче класи.

Клас NThread призначений для створення визначеної користувачем кількості паралельних потоків програми, їх виконання до кінця та обробки результатів цих потоків. Цей клас моделює метод багатокористувацького запиту.

Класи різних типів діаграм будуються з урахуванням того, що будь-яка з них може бути використана для виведення. Тому всі вони повинні реалізувати інтерфейс ChartProducer . Цей інтерфейс описує метод createChart (), який мають реалізувати всі класи діаграм. У цьому методі створюються параметри відображення графіка.

Класи ChartEngine і ChartDescriptor призначені для аналізу файлу конфігурації діаграми . xml . За допомогою цих класів визначаються початкові параметри відображення кожного типу графа, реалізованого в системі. Файл конфігурації діаграми . _ xml має визначати існуючі типи діаграм і зберігати початкові параметри розмірів діаграм.

Класи ParseData та StatisticData призначені для аналізу файлу журналу, створеного програмою аналізу трафіку BWMmeter . Вони застосовують аналіз рядків файлу журналу для визначення типу переданих даних (вхідний або вихідний трафік), а потім формують часовий ряд для відображення у формі графіка за допомогою класів MyChart , MyChart 2, MyChart 3, MyChart 4, MyChart 5 тощо . .

Висновки розділу 3

У цій частині проведено практичну реалізацію розроблених методів та алгоритмів, здійснено впровадження програмного забезпечення.

ВИСНОВКИ

Основним результатом магістерської роботи є розробка та обґрунтування методу «прозорих журналів» для організації базового процесу тестування систем обробки веб- даних в умовах обмежених ресурсів, а також удосконалення окремих методів, спрямованих на підвищення якості програмного забезпечення.

Методологія, запропонована в магістерській роботі, поєднує стратегії інтеграційного тестування (тестування модулів/компонентів в одній програмі з використанням підходів, властивих функціональному тестуванню, тобто методу «сірого ящика») та системного тестування (тестування зовнішніх інтерфейсів або інтерфейсів користувача).

Записи журналу можна використовувати скільки завгодно разів. Набір записів можна перетворити в будь-який формат представлення, який можна використовувати в інших методах автоматизації процесу тестування, наприклад, за допомогою мереж Петрі.

Записи журналу "прозоре журналювання" для різних компонентів програми (читання/запис). На основі записів із журналу неможливо виконати тести, які підтверджують точність вмісту веб-сторінки з точки зору користувача. Це треба робити іншим способом.

Впровадження результатів дослідження в організації розробника Центру інформаційних технологій Тернопільського національного економічного університету дозволило підвищити ефективність процесу тестування, зробити його відчутним і простежуваним, а також значно зменшити кількість дефектів у системі. розроблений.

СПИСОК ЛІТЕРАТУРИ

1. Сергієнко І. В. Становлення та розвиток досліджень інформатики. – К.: Наукова думка. – 1998. - 205 с.

2. Принципи інжинірингу якості програмних систем / Ф.І.Андон, Г.І.Ковал, Т.М.Коротун, В.Ю. Суслов / За ред. І. В. Сергієнко. – К.: Академперіодика. - 2002. - 504 с.

3. Парадигма якості ПЗ / Андон Ф.Ю., Суслов В.Ю., Коротун Т.М., Коваль Г.І. // Проблеми програмування. -1999 рік. -№2. – С. 51-62.

4. Липаєв В. В. Забезпечення якості програмного забезпечення. – М: СИНЕРГ. - 2001. – 380 с.

5. ДСТУ 3918-99 Інформаційні технології. Процеси життєвого циклу програмного забезпечення. – Київ. - Держстандарт України. – 2000. – 49 с.

6. Коваль Г.Ю., Коротун Т.М., Остапенко А.П. Профілактичне тестування та оцінка надійності програмного забезпечення як форма управління ризиками проекту. // субота. Розробка програмного забезпечення. – Київ. – 1993. – С. 19-26.

7. Мороз Г.Б., Коваль Г.І., Коротун Т.М. Інженерні цілі та задачі визначення надійності програмного забезпечення // Проблеми програмування. - 1997. - Ні. 1. – С. 98-106.

8. Управління ризиками програмних проектів / Андон Ф.І. Суслов В.Ю. Коротун Т. М., Коваль Г. І. Слабоспицька О. А. // Проблеми програмування. - 1999. - Ні. 1. – С. 53-62.

9. Лаврищева Є.М., Коротун Т.М. Побудова процесу тестування програмної системи // Проблеми програмування.-2002. - Ні. 1-2. – С. 272-281.